

Approximate Maximum Flow on Separable Undirected Graphs ^{*}

Gary L. Miller
CMU
glmiller@cs.cmu.edu

Richard Peng [†]
CMU
yangp@cs.cmu.edu

October 19, 2012

Abstract

We present faster algorithms for approximate maximum flow in undirected graphs with good separator structures, such as bounded genus, minor free, and geometric graphs. Given such a graph with n vertices, m edges along with a recursive \sqrt{n} -vertex separator structure, our algorithm finds an $1 - \epsilon$ approximate maximum flow in time $\tilde{O}(m^{6/5} \mathbf{poly}(\epsilon^{-1}))$, ignoring poly-logarithmic terms. Similar speedups are also achieved for separable graphs with larger size separators albeit with larger run times. These bounds also apply to image problems in two and three dimensions.

Key to our algorithm is an intermediate problem that we term grouped L_2 flow, which exists between maximum flows and electrical flows. Our algorithm also makes use of spectral vertex sparsifiers in order to remove vertices while preserving the energy dissipation of electrical flows. We also give faster spectral vertex sparsification algorithms on well separated graphs, which may be of independent interest.

1 Introduction

The maximum flow problem is a fundamental algorithmic question introduced by Ford and Fulkerson [FF56] and studied extensively since. In its simplest form, the problem asks for a flow \mathbf{f} in a graph G that routes the maximum amount from a source s to a sink t , subject to the constraint that the amount of flow on each edge does not exceed its capacity.

Due to its origins in studying railroad network problems [HR56], the maximum flow problem on planar graphs has also received considerable attention. The Ford-Fulkerson paper gives an algorithm for the case where s and t lie on the same face [FF56]. In this setting a long line of work [Has81, JV82, MN95] led to a nearly-linear time algorithm on directed planar graphs [BK09]. In the case of undirected graphs, which our work addresses, the running time has been brought to as low as $O(n \log \log n)$ [INSWN11]. While originally intended for transportation problems, these algorithms also have a wide range of applications in computer vision [Bor08].

Many applications contain a number of irregularities that are either intrinsic or added by the user. These irregularities are sufficiently common to be studied in early works on planar separators by Lipton and Tarjan [LT79] (e.g. a 2D finite element graphs where one connects any two vertices that share a face). They can take the form of bridges in transportation problems, or extra correlations in non-local methods for images [BCM08]. As a result, algorithms resilient to a small number of edge changes are needed to handle many of these instances. Progress in this direction was made by Chambers et al. [CEN09b, CEN09a], who gave an algorithm for s - t maximum flow on a genus g graph with integer capacities summing up to C that runs in $O(\mathbf{poly}(g)n \log^2 n \log^2 C)$ time.

^{*}Partially supported by the National Science Foundation under grant number CCF-1018463.

[†]Supported by a Microsoft Fellowship

It is also possible for small modifications to introduce highly connected minors while preserving the overall structure of the graph. For example, two or more layers of an $O(\sqrt{n}) \times O(\sqrt{n})$ grid with corresponding vertices connected on the layers is similar to the (planar) square grid, yet has a large $K_{\sqrt{n}}$ minor. Graphs similar to this often occur in computer vision problems involving 3-D images, or 2-D videos [VSN09, LF10]. As a result, we believe that a more robust algorithm can be directly used for a considerably wider range of applications.

In this work we introduce a novel approach for computing approximate maximum flow on these graph families. We show that good separator structures alone are sufficient for faster algorithms for computing a flow within $1 - \epsilon$ of the maximum flow. Our work is based on the multiplicative weights method recently used by Christiano et al. [CKM⁺11] in their faster approximate maximum flow algorithm. We develop a two-level, recursive approach that gives even faster algorithms on separable undirected graphs. To our knowledge, the only previous work using just separator structures was by Imai and Iwano [II90]. Their algorithm takes advantage of the separators by speeding up the L_2 minimizations, giving a running time of about $O(m^{1.59})$. As a maximum flow algorithm, however, their result has been superceded by the result on general graphs by Goldberg and Rao [GR98].

The Christiano et al. algorithm uses very different techniques than previous work. Their algorithm constructs an approximate maximum flow by taking the average of a sequence of electrical flows, each constructed based on previous ones. These electrical flows are in turn computed using the nearly-linear time solvers for SDD linear systems introduced by Spielman and Teng [ST04, ST06, KMP10a, KMP11b]. The solver algorithm can be viewed as recursion on a hierarchy of graphs, each increasingly sparser than the previous one. Direct adaptations of this type of recursion on gradually shrinking graphs have yielded fast approximate cut algorithms [M10], albeit at the cost of some loss in the quality of the solution. This work naturally leads to the question of whether the Christiano et al. algorithm can be combined with some type of recursive routine to get faster algorithms with smaller errors.

In a work joint with Hui Han Chin and Aleksander Mądry [HMMP12] that addressed problems from image processing, we observed that a coarser grouping of edges leads to an intermediate problem between maximum flow and electrical flow. This problem, which we term the **grouped L_2 flow** problem, has two key properties that we'll show. It can be used in place of electrical flows in the Christiano et al. algorithm and lead to fewer iterations when the groups are small. Furthermore, on well separated graphs, these problems can be approximated in nearly-linear time for intermediate values of k . Combining the decreased iteration count with the nearly-linear running time of each iteration gives a faster algorithm. Our algorithm can also be readily extended to many of the image processing related tasks, such as isotropic total variation minimization [GY04]. Due to the additional components of these objectives and the readiness of the extension, we restrict our presentation to only approximating the maximum flow.

Our algorithm constructs the intermediate grouped flow problem based on a r -division, which are partitions of the graph into n/r groups each with small boundary of size $O(\sqrt{r})$. Then we show that vertices incident to edges from only one group can be removed using **spectral vertex sparsifiers**. This removal obtained by modifying the generalized nested dissection algorithm by Lipton, Rose and Tarjan [LRT79]. This result allows us to reduce the problem size by a factor of $\Omega(\sqrt{r})$, leading to savings in the cost of computing these flows. When given the separator structures generated by algorithms such as [LT79, KR10, WN11], our main result gives:

Theorem 1.1 *Given a graph $G = (V, E)$ with n vertices and m edges with vertices partitioned into a r -division along with recursive separator trees for each group, capacities $\mathbf{u} \in \mathbb{R}^E$ and source/sink vertices s and t , we can compute a $1 - \epsilon$ approximate maximum $s - t$ flow in $\tilde{O}(m^{6/5} \epsilon^{-6.1})$ ¹ time.*

Our results are applicable to a variety of graph families where separators can be found efficiently.

¹We use $\tilde{O}(f(m))$ to denote $\tilde{O}(f(m) \log^c f(m))$ for some constant c .

Families where nearly-linear time separator algorithms are known include planar [LT79], bounded genus, h -minor free for fixed h [KR10] and nearest neighbor geometric graphs [MTTV97]. For h -minor free graphs with larger values of h (e.g. n^ϵ), speedups can still be obtained using the algorithm by Wulff-Nilsen [WN11], which runs in $\tilde{O}(n^{5/4+O(\epsilon)})$ time.

Aside from giving the first speedup over approximate general flow algorithms for the class of minor-free and shallow minor-free graphs, our weaker requirement of separator structure have several other advantages. The requirement of a top-level partition into n/r pieces naturally incorporate $O(n/r)$ extra edges. When an even larger number of such extra edges are present, the performance of our algorithm can degrade smoothly to that of Christiano et al. algorithm by a suitable change of parameters. Similar modifications can also be done if the separators given are of size n^β instead of \sqrt{n} . If $\beta < 2/3$, a modified version of our algorithm still gives running times faster than $\tilde{O}(m^{4/3} \text{poly}(\epsilon^{-1}))$.

This robustness of our algorithm means it extends naturally to multiple sources and sinks, which occur in a variety of applications. On general graphs, this can be handled by introducing a super source and super sink, then connecting all vertices to them with appropriate capacities. However, this modification can significantly change the connectivity structure. As a result this extension on planar graphs has been studied as an important problem on its own [MN95], with a nearly-linear time algorithm being a recent result [BKM⁺11]. The milder requirement of partitions and separators on the other hand can naturally incorporate a super source and sink. These two vertices can be incorporated as part of every partition with only additive constant overhead. This means the guarantees in Theorem 1.1 includes the multiple sources/sinks case, and may have other advantages in terms of robustness.

Our results can also be combined with graph smoothing and sampling techniques [Kar98] to obtain speedups by a factor of about $(m/n)^{1/5}$. As planar, bounded genus, and minor-free graphs have $m = O(n)$, this improvement is only applicable for dense graphs that have good underlying vertex separator structures, and we omit it from our presentation.

The paper is organized as follows: Section 2 gives formal definitions of the tools we use. The main components of our algorithm and their interactions are described in Section 3. The analysis of these components are direct adaptations of known results, with error analyses incorporated appropriately. They are given in Sections 4, 5 as well as Appendices A, B. Possible extensions of our results are discussed in Section 6.

2 Background and Notations

In this section we review some of concepts and notation used in the paper. We use subscripts to distinguish between variants of variables and (\cdot) to denote entries in vectors/matrices.

2.1 Graphs, Graph Laplacians and Spectral Ordering of Matrices

We use $G = (V, E)$ to denote an undirected graph. If we orient each edge $e = uv$ arbitrarily, the edge-vertex incidence matrix of G is defined as:

$$\mathbf{B}(e, u) = \begin{cases} -1 & \text{if } u \text{ is the head of } e \\ 1 & \text{if } u \text{ is the tail of } e \\ 0 & \text{otherwise} \end{cases}$$

Given edge weights of G , \mathbf{w} , we can define the Laplacian of G as $\mathbf{L} = \mathbf{B}^T \mathbf{W} \mathbf{B}$, where \mathbf{w} is the diagonal matrix of edge weights. Entry-wise we have that $\mathbf{L}(u, v)$ is the degree of vertex u if $v = u$; $-\mathbf{w}_{uv}$ if uv is an edge; and 0 otherwise. Many of our routines have logarithmic dependency on the ratio between the maximum and minimum entry of \mathbf{w} . In all of their invocations needed for our main result, this parameter is bounded by $\text{poly}(n)$. As a result, this term can be viewed as an additional factor of $\log n$ and is not crucial to understanding our approach. In the formal presentation of our results, we will use $U(\mathbf{w})$ to denote $\max(\mathbf{w})/\min(\mathbf{w})$, and will also apply this notation to other vectors.

We will use \mathbf{L} interchangeably with G . When the underlying graph is not clear from the context, we will also denote it using $\mathbf{L}(G)$. It can be shown that graph Laplacians are positive semi-definite. A standard notation that we'll use repeatedly throughout our presentation for comparing matrices is \preceq . Specifically, if \mathbf{A} and \mathbf{B} are positive semi-definite matrices, we use $\mathbf{A} \preceq \mathbf{B}$ to denote that $\mathbf{B} - \mathbf{A}$ is positive semi-definite. Or in other words, $\mathbf{x}^T \mathbf{A} \mathbf{x} \leq \mathbf{x}^T \mathbf{B} \mathbf{x}$ for all vector \mathbf{x} . We will also use $0 = \lambda_1(\mathbf{L}) \leq \lambda_2(\mathbf{L}) \leq \dots \leq \lambda_n(\mathbf{L})$ to denote the n eigenvalues of \mathbf{L} .

2.2 Approximate Maximum Flow

If we're also given capacities on the edges of G , $\mathbf{u} : E \rightarrow \mathbb{R}^+$. Then for a flow on G , $\mathbf{f} : E \rightarrow \mathbb{R}^+$, we can define the (undirected) **congestion** of this flow w.r.t. edge e as:

$$\text{cong}(\mathbf{f}, e) = \frac{|\mathbf{f}(e)|}{\mathbf{u}(e)}$$

In an undirected graph, the requirement that a flow obeys capacities can be expressed as $\text{cong}(\mathbf{f}, e) \leq 1$. We would also like the flow to meet certain demands at vertices. Specifically given a demand vector \mathbf{d} that sums to 0, this constraint can be written as $\mathbf{B}^T \mathbf{f} = \mathbf{d}$. Since we're allowed to add a super source/sink, one demand vector of particular interest is \mathbf{d} being only non-zero at two vertices, s and t . This corresponds to the s - t maximum flow problem, which seeks to maximize the amount of flow entering t (or equivalently, exiting s) while obeying capacities and routing the same amount in and out of all other vertices. The approximate maximum flow problem in turn becomes finding a flow that routes at least $1 - \epsilon$ of the maximum amount. As flows on connected components behave independently of each other, we may also assume that G is connected.

2.3 Electrical Flows

Given a set of resistive values on the edges, $\mathbf{r} : E \rightarrow \mathbb{R}^+$, the electrical energy dissipation of a flow \mathbf{f} , $\mathcal{E}_{\mathbf{f}}(\mathbf{r})$, can be defined as:

$$\mathcal{E}_{\mathbf{f}}(\mathbf{r}) = \sum_e \mathbf{r}(e) \mathbf{f}(e)^2$$

The set of resistances also defines a natural graph Laplacian, namely $\mathbf{L} = \mathbf{B}^T \mathbf{R}^{-1} \mathbf{B}$. For a demand \mathbf{d} , let $\bar{\mathbf{f}}$ denote the flow that minimizes the electrical energy and let $\mathcal{E}(\mathbf{r}) = \mathcal{E}_{\bar{\mathbf{f}}}(\mathbf{r})$. We use the following facts, shown in Section 2.3.1 of [CKM⁺11], about the energy dissipation of the optimum electrical flow.

Lemma 2.1

$$\mathcal{E}(\mathbf{r}) = \mathbf{d}^T \mathbf{L}^+ \mathbf{d}$$

Where \mathbf{d} is the column demand vector, and \mathbf{L}^+ is the pseudo-inverse of \mathbf{L} .

A more thorough treatment of various properties of electrical flows and associated voltage values can be found in [DS84]. In order to compute electrical flows, we invoke the following result about computing almost-optimal electrical flows using SDD linear system solvers

Theorem 2.2 *There is an algorithm ELECTRICALFLOW such that when given a weighted graph $G = (V, E, \mathbf{r})$, along with any demand vector \mathbf{d} such that the corresponding minimum energy flow is $\bar{\mathbf{f}}$. ELECTRICALFLOW computes in $\tilde{O}(m \log(U(\mathbf{r})\delta^{-1}))$ time a flow $\tilde{\mathbf{f}}$ such that*

1. $\tilde{\mathbf{f}}$ meets the demands $\mathbf{B}^T \tilde{\mathbf{f}} = \mathbf{d}$

2. $\mathcal{E}(\tilde{\mathbf{f}}) \leq (1 + \delta)\mathcal{E}(\mathbf{r})$
3. For every edge e , $\sum_e |\mathbf{r}(e)\tilde{\mathbf{f}}(e)^2 - \mathbf{r}(e)\tilde{\mathbf{f}}(e)^2| \leq \delta\mathcal{E}_{\tilde{\mathbf{f}}}(\mathbf{r})$

A more detailed discussion of electrical flows, and its natural dual, effective resistances, can be found in Section 2 of [CKM⁺11].

2.4 Graph Separators

Our algorithm makes extensive use of separator structures, both as r -divisions and as recursive separator trees. A single vertex separator can be formalized as:

Definition 2.3 *On a graph G with vertex set V , a subgraph S is an α -separator if $G \setminus S$ can be partitioned into two graphs $G^{(1)}$ and $G^{(2)}$ such that there are no edges between them and $|V(G^{(1)})|, |V(G^{(2)})| \leq \alpha|V(G)|$.*

There is a rich literature on efficiently finding such separators for planar, bounded genus, minor free, and geometric graphs [LT79, MTTV97, KR10, WN11]. Our algorithms do not change the connectivity of the graph, and use the separators repeatedly. As a result, we treat structures related to separators as separate entities computed before running our algorithms.

Given a subgraph G' of G , we let its **vertex boundary** $V_{bdry}(G')$ be vertices that have edges to $V \setminus V(G')$ and denote $V(G') \setminus V_{bdry}(G')$ as $V_{intr}(G')$. We can now define r -divisions, which are multi-way partitions of $V(G)$.

Definition 2.4 *A r -division of a graph $G = (V, E)$ is a partition of it into groups $G^{(1)} \dots G^{(k)}$ such that each edge belongs to exactly one group and:*

- The number of groups, k is at most $O(\frac{n}{r})$
- $|E(G^{(i)})| \leq r$
- The number of boundary vertices of each group, $V_{bdry}(G^{(i)})$ has size at most $O(\sqrt{r})$.

Fredrickson [Fre87] showed that with an extra $\log n$ factor in running time, one could compute separators repeatedly to obtain a r -division of the graph into pieces of size $O(r)$. In the planar case this was further improved to linear time by Goodrich [Goo95].

One of our algorithms for finding sparser equivalents of each group makes use of a recursive separator trees. These are obtained by recursively computing separators on the components, leading to a hierarchical partitioning of the graph. These structures were first used in nested-dissection algorithms [GT87, LRT79]. A subtle, but significant distinction exists in their construction. For planar, or more generally sparse-contractible graphs, it was shown that building separators on each component suffices for a fast algorithm [GT87]. However, if we are to only assume separator based properties, we need also keep the separator itself in one of the components. This ensures that the separator itself is partitioned in future layers, and won't eventually become a bottleneck in the algorithm. Lipton, Rose and Tarjan [LRT79] showed in their generalized nested dissection algorithm that these more carefully constructed separator trees alone are sufficient for good guarantees. The separator trees that we rely on are based on their construction and is described below.

Definition 2.5 *A recursive α -separator structure $\mathcal{S}(\bar{G})$ for a graph $\bar{G} = (V, E)$ is a tree where each node corresponds to a subgraph and satisfies the following properties:*

- The root node equals to \bar{G} .

- Each leaf node G has $|V(G)| \leq C$, where C is an absolute constant.
- Each non-leaf node corresponding to graph G has a α -separator S . Furthermore its two children corresponds to graphs induced on $G^{(1)} \cup S$ and $G^{(2)} \cup S$ with edges in S belonging to both.

Theorem 1 of [LRT79] showed that finding the full separator tree can also be done with an extra $\log n$ factor overhead in the running times of computing separators.

2.5 Schur Complement

The partition of V into V_{intr} and V_{bdry} naturally allows us to partition \mathbf{L} into 4 blocks:

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_{intr} & \mathbf{L}_{mid} \\ \mathbf{L}_{mid}^T & \mathbf{L}_{bdry} \end{bmatrix}$$

Note that by our assumption of G being connected, \mathbf{L}_{intr} is invertible when both V_{bdry} and V_{intr} are non-empty. The Schur complement acts as a smaller-sized equivalent of \mathbf{L} on only the boundary vertices.

Definition 2.6 *The Schur complement of a graph Laplacian \mathbf{L} with respect to a set of boundary vertices V_{bdry} , $\text{SCHURCOMPLEMENT}(\mathbf{L}, V_{bdry})$ is:*

$$\mathbf{L}_{bdry} - \mathbf{L}_{mid}^T \mathbf{L}_{intr}^{-1} \mathbf{L}_{mid}$$

We will also label objects associated with the Schur complement with the subscript $_{schur}$. Specifically, the Schur complement itself will also be denoted using \mathbf{L}_{schur} .

3 Overview of Our Algorithms

We present a summary of our results in this section. The bounds for our algorithms are probabilistic due to calls to spectral sparsifiers [SS08] that randomly sample graphs. It was shown in [KL11] that with constant factor slowdown in running time, these routines can succeed with high probability. Therefore we omit the failure probability in the guarantees of our algorithms to simplify our presentation. An example of applying our algorithm to an instance on a square mesh is given at the end of this section, with illustrations given in Figure 1. The running time of various components are also given in Figure 2 in Section 6. It might be helpful to refer to this example, for intuition on the purposes of each component of our algorithm.

The Christiano et al. algorithm [CKM⁺11] showed that electrical flows can be combined to form a flow that approximately meets all the edge capacities. This connection between electrical and maximum flows can be illustrated by a closer look at the objective functions of these two flows. Maximum flow imposes capacity constraints of $\mathbf{f}(e) \leq \mathbf{u}(e)$ on each edge, which are equivalent to $(\mathbf{f}(e)/\mathbf{u}(e))^2 \leq 1$. The weights in the (ϵ, ρ) oracle, and in turn electrical flows can be viewed as a way to create a single constraint from all m edges, in the form of $\sum_e \mathbf{w}(e)(\mathbf{f}(e)/\mathbf{u}(e))^2 \leq \sum_e \mathbf{w}(e)$.

Instead of aggregating these into a single constraint, we can aggregate over k groups of edges, leading to an intermediate problem. The congestion of an edge can then be generalized to that of a group of edges, and we define $\mathbf{cong}(\mathbf{f}, i)$ as the square root of the total energy among edges in the i -th group, $G^{(i)}$:

$$\mathbf{cong}(\mathbf{f}, i) = \sqrt{\sum_{e \in E(G^{(i)})} \mathbf{w}(e) \mathbf{f}(e)^2}$$

Then given a demand vector \mathbf{d} , an instance of **grouped L_2 flow** can be formalized as minimizing the maximum congestion over the k groups.

Definition 3.1 A **grouped L_2 flow problem** is, given a graph $G = (V, E)$ and edge weights \mathbf{w} , source/sink pair s and t , along with a partition of the edges into S_1, \dots, S_k , find a flow \mathbf{f} that meets the demands, $\mathbf{B}^T \mathbf{f} = \mathbf{d}$. The congestion of this flow is the maximum congestion over all groups, $\max_i \text{cong}(\mathbf{f}, i)$.

This problem is also the natural dual of the grouped least squares problem introduced in [HMMP12], which generalizes a variety of objective functions from image processing. We say that an algorithm approximates grouped L_2 flows if given a graph where there exists a flow \mathbf{f} with $\text{cong}(\mathbf{f}, i) \leq 1 - \epsilon$, it returns a grouped L_2 flow with $\text{cong}(\mathbf{f}, i) \leq 1 + \epsilon$, and possibly “fail” otherwise. We show in Section 4 that if each group has at most r edges, the approximate maximum flow problem can be solved using a sequence of about $r^{1/2}$ approximate grouped L_2 flow problems.

Lemma 3.2 Suppose we have access to a routine for approximating grouped L_2 flows, APPROXGROUPEDFLOW. If $G^{(1)}, G^{(2)} \dots G^{(k)}$ satisfy $|G^{(i)}| \leq r$ and $\epsilon < 1/2$, an $1 - \epsilon$ approximate maximum flow can be computed by $\tilde{O}(r^{1/2} \epsilon^{-1/2})$ calls to APPROXGROUPEDFLOW. Furthermore, in each of these calls, the edge weights \mathbf{w} satisfy $U(\mathbf{w}) \leq O(m^3 \epsilon^{-3})$.

The rest of our algorithm can then be viewed as giving a series of faster algorithms for approximating grouped L_2 flows. We first show that the group L_2 flows can be approximated in $O(k^{1/3})$ iterations, each involving a SDD linear system solve. This is done by modifying the algorithms from [CKM⁺11, HMMP12]. This algorithm and its analysis are described in Appendix A. Note that the number of iterations only depends on k , the number of groups. Also, this algorithm returns “fail” when it can certify that a grouped flow with small congestion does not exist. As a result, our approximate max-flow algorithm, as stated, only produces a flow. We show in Appendix D that it can be modified to return a cut of small value instead.

Theorem 3.3 Given a graph $G = (V, E, \mathbf{w})$ with edges partitioned into k groups $S_1 \dots S_k$, along with an edge weight vector $\mathbf{w} : E \rightarrow \mathbb{R}^+$ and a demand vector $\mathbf{d} : V \rightarrow \mathbb{R}$. If there exists a flow $\tilde{\mathbf{f}}$ that meets the demands and satisfies $\text{cong}(\tilde{\mathbf{f}}, i) \leq 1 - \epsilon$, then GROUPEDFLOW($G, \mathbf{d}, \mathbf{w}, \epsilon/10$) returns in $\tilde{O}(mk^{1/3} \log(U(\mathbf{w})) \epsilon^{-8/3})$ time a flow $\tilde{\mathbf{f}}$ where for all groups we have: $\text{cong}(\tilde{\mathbf{f}}, i) \leq (1 + \epsilon)$.

We can combine this algorithm directly with Lemma 3.2 by grouping edges into $k = O(m/r)$ groups of size r . This leads to a total running time of:

$$\begin{aligned} & \tilde{O}(mk^{1/3} \epsilon^{-8/3}) \cdot \tilde{O}(r^{1/2} \epsilon^{-5/2}) \\ &= \tilde{O}(m(m/r)^{1/3} r^{1/2} \epsilon^{-31/6}) \\ &= \tilde{O}(m^{4/3} r^{1/6} \epsilon^{-31/6}) \end{aligned}$$

As $r \geq 1$, this cannot lead to an algorithm that runs faster than $\tilde{O}(m^{4/3})$. In order to obtain speedups, we need to reduce the m term in the computation of grouped flows. This term corresponds to the number of edges in the linear systems solved in each of the $\tilde{O}(k^{1/3} \epsilon^{-8/3})$ iterations needed for grouped flow computation. We do so using the property that each group’s boundary is smaller than its size by a factor of about $r^{1/2}$. For each subgraph $G^{(i)}$, we construct a smaller ‘equivalent’ of it that only contains vertices of $V_{\text{bdry}}(G^{(i)})$.

The Schur complement described in Definition 2.6 is a natural way to find such a smaller graph on boundary vertices. However, the exact Schur complement is usually a dense graph with $(r^{1/2})^2 = r$ edges. We obtain further reductions by showing that for approximating grouped flows, a graph that’s spectrally close to the Schur complement suffices. This allows us to use the tool of spectral sparsification [SS08], which generates sparse approximations to such dense graphs. Formally, we denote such approximations to the Schur complements spectral vertex sparsifiers:

Definition 3.4 Given a graph G and a set of boundary vertices $V_{\text{bdry}}(G)$. A graph \tilde{G}_{schur} on vertices $V_{\text{bdry}}(G)$ with corresponding Laplacian $\tilde{\mathbf{L}}_{\text{schur}}$ is an ϵ -spectral vertex sparsifier for $(G, V_{\text{bdry}}(G))$ if:

1. \tilde{G}_{schur} has $\tilde{O}(|V_{\text{bdry}}(G)|\epsilon^{-2})$ edges.
2. \tilde{G}_{schur} and $\text{SCHURCOMPLEMENT}(\mathbf{L}, V_{\text{intr}})$ are spectrally close.

$$(1 - \epsilon)\mathbf{L}_{\text{schur}} \preceq \tilde{\mathbf{L}}_{\text{schur}} \preceq (1 + \epsilon)\mathbf{L}_{\text{schur}}$$

3. $U(\mathbf{w}(\tilde{G}_{\text{schur}})) \leq \text{poly}(n)U(\mathbf{w}(G))$.

Our speedup is obtained by solving the grouped L_2 flow problem on a graph with all groups replaced by their **spectral vertex sparsifiers**. The conversion of flows to and from the smaller graph is analyzed in Section 5. It leads an algorithm APPROXGROUPEDFLOW whose running time is about $\tilde{O}(m^{4/3}r^{-5/6} + m)$. This can be viewed as depending inversely on r , the size of the groups, or being proportional to the number of groups.

Theorem 3.5 Let $G = (V, E, \mathbf{w})$ be a graph where there exists a grouped L_2 flow meeting demand \mathbf{d} with maximum congestion $1 - \epsilon$. Given a r -division of G and $\epsilon/10$ -spectral vertex sparsifiers for all the groups, APPROXGROUPEDFLOW returns in $\tilde{O}(m^{4/3}r^{-5/6}\epsilon^{-14/3} \log(U(\mathbf{w})/\epsilon) + m \log(U(\mathbf{w})/\epsilon))$ time a flow meeting demands \mathbf{d} where each group has congestion at most $1 + \epsilon$.

Note that when $r > O(m^{2/5})$, this total running time is about $\tilde{O}(m)$. Therefore, running time bottleneck now moves to the cost of computing spectral vertex sparsifiers of the $O(m/r)$ groups, each with $O(r)$ edges and $O(r^{1/2})$ boundary vertices. In the rest of this section, We outline our spectral vertex sparsification algorithms. They're consequences of known results of using high quality approximate solvers instead of exact inverses, applied both once or recursively within the generalized nested dissection framework. Details and proofs on them are given in Appendix B.

A direct approach is to compute the Schur complement as Definition 2.6, with sparsification done both before and after computing the Schur complement. Since \mathbf{L}_{intr} is diagonally dominant, we can invoke SDD linear system solvers on \mathbf{L}_{intr} with on each column of \mathbf{L}_{mid} . This gives an almost-exact approximation to $\mathbf{L}_{\text{intr}}^{-1}\mathbf{L}_{\text{mid}}$ in about $\tilde{O}(r^{1.5})$ time. Multiplying $\mathbf{L}_{\text{mid}}^T$ against this $\tilde{O}(r^{1/2}) \times \tilde{O}(r^{1/2})$ matrix takes $\tilde{O}(r^{3/2})$ time as \mathbf{L}_{mid} has $\tilde{O}(r)$ non-zero entries. Analyzing the error incurred from the solver and applying spectral sparsification at the end leads to a one-step construction of the spectral vertex sparsifier:

Lemma 3.6 Given a graph $G = (V, E, \mathbf{w})$ with n vertices and m edges and error bound $0 < \epsilon < 1/2$. There is a routine ONESTEPVERTEXSPARSIFY that returns in $\tilde{O}(m + |V_{\text{bdry}}(G)|n\epsilon^{-2} \log(U(\mathbf{w})))$ time an ϵ -spectral vertex sparsifier for $(G, V_{\text{bdry}}(G))$, \tilde{G}_{schur} , such that $U(\mathbf{w}(\tilde{G}_{\text{schur}})) \leq O(m^5 U(\mathbf{w}))$.

Combining this with Theorems 3.5 and 3.3 gives a total running time of $\tilde{O}(mr\epsilon^{-9/2} + m^{4/3}r^{-1/3}\epsilon^{-43/6})$. This is minimized when $r = m^{1/4}\epsilon^{-2}$, which in turn gives a total of $\tilde{O}(m^{5/4}\epsilon^{-13/2})$.

This algorithm is applicable when we can find, or are given a good partition of our initial graph, but cannot do so repeatedly in the rest of the graph. For example, for shallow minor-free graphs (which are not guaranteed to have recursive separator structures), the result of [WN11] allows us to find the top level r -division in $\tilde{O}(n^{5/4})$ time, leading to an approximate maximum flow algorithm that runs in $\tilde{O}(m^{5/4}\epsilon^{-6})$ time for shallow minor free graphs. In the more general case where each group of the r -division has a larger boundary of size r^β , this type of runtime savings still hold. However, the gains stop at $\beta = 2/3$ which corresponds to 3D cubes.

For graphs which are recursively separable, we can take advantage of the existence of a full separator tree to speed up the computation of the spectral vertex sparsifier. Specifically we an algorithm analogous

to parallel nested dissection first introduced by Pan and Reif [PR93]. This approach ensures that every time we invoke the one-step algorithm given in Lemma 3.6, the boundary and interior are of comparable size. Interleaving sparsification at all stages with higher accuracy requirements leads to an algorithm that runs in $\tilde{O}(r)$ time.

Lemma 3.7 *Let $G = (V, E, \mathbf{w})$ be a graph with n vertices and m edges given along with a $9/10$ -separator tree \mathcal{S} . Given a partition of the vertices V into boundary and interior vertices such that $|V_{\text{bdry}}(G)| \leq O(n^{1/2})$, and an error parameter ϵ , VERTEXSPARSIFY($\mathcal{L}, V_{\text{bdry}}(G), \epsilon/2 \log_{20/19} n$) returns in $\tilde{O}(m + n\epsilon^{-2} \log U(\mathbf{w}))$ time an ϵ -spectral vertex sparsifier for $(G, V_{\text{bdry}}(G))$, \tilde{G}_{schur} , such that $U(\mathbf{w}(\tilde{G}_{\text{schur}})) \leq O(m^5 U(\mathbf{w}))$.*

This leads to a faster algorithm that finds a $1 - \epsilon$ approximate maximum flow in $\tilde{O}(mr^{1/2}\epsilon^{-9/2} + m^{4/3}r^{-1/3}\epsilon^{-43/6})$ time on well-separated graphs. This is minimized when $r = m^{2/5}\epsilon^{-16/5}$, giving a total running time of $\tilde{O}(m^{6/5}\epsilon^{-61/10})$ and in turn Theorem 1.1.

In order to give an overall picture of our algorithm, it's helpful to consider the case where we're trying to route flows from the top left to bottom right on a $n^{1/2} \times n^{1/2}$ square mesh. The grid structure is chosen for this example in order to simplify the description of the r -division. Of course, in this case the faster planar maximum flow algorithms can be used instead. However, all the steps are essentially the same for separable graphs.

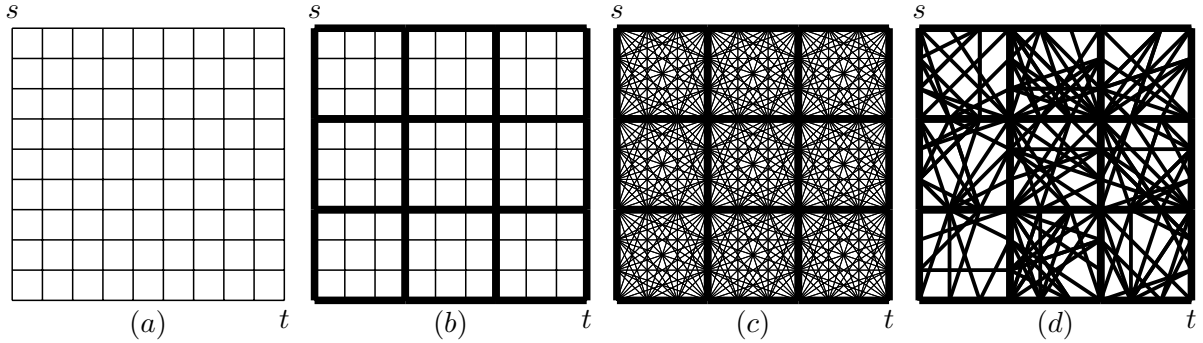


Figure 1: Illustrations of the main structures that occur in our algorithm from left to right (a) original graph; (b) grid partitioned into r -divisions; (c) exact dense spectral vertex sparsifier; (d) sparse spectral vertex sparsifier.

We use this example to walk through the main steps of our algorithm on well-separated graphs. Given the input graph shown in Figure 1 (a), we decompose it into graphs with $r = O(n^{2/5})$ vertices, of which $O(r^{1/2}) = O(n^{1/5})$ are on the boundary, (in this case, smaller squares with side length $O(n^{1/5})$). This leads to $k = n/r = O(n^{3/5})$ groups, giving the graph shown in (b). To find an approximate grouped L_2 flow, we construct spectral vertex sparsifiers using Lemma 3.6 or 3.7. These graphs have both fewer edges and vertices, giving (d). An exact way of doing this sparsification is to remove all internal nodes of each group using partial Gaussian elimination, giving (c) where each group only has the $O(r^{1/2}) = O(n^{1/5})$ boundary vertices. However, these groups are now dense graphs and can have $O((r^{1/2})^2) = O(r) = O(n^{2/5})$ edges per group. However, the combination of this and spectral sparsification [SS08] gives $\tilde{O}(r^{1/2}) = \tilde{O}(n^{1/5})$ vertices and edges per group. From the perspective of grouped L_2 flow problem (b), (c), and (d) are roughly equivalent except that (d) is substantially smaller. Our algorithm then solves an approximate maximum flow problem on the grid (a) by averaging a sequence of grouped flows. The guarantees given in Lemma 3.2 leads to about $\tilde{O}(r^{1/2}) = \tilde{O}(n^{1/5})$ grouped L_2 flows on graph (b). Each such instance is solved using the algorithm APPROXGROUPEDFLOW described in Theorem 3.5. For each grouped L_2 flows

on graph (b) we construct the smaller but equivalent instance (d) as described above in $\tilde{O}(n)$ time. We then solve the grouped L_2 problems on (d) using Theorem 3.3, leading to a total running time of about $\tilde{O}(n^{4/3}r^{-5/6} + n) = \tilde{O}(n)$. This running time can be further broken down into about $\tilde{O}(k^{1/3}) = \tilde{O}(n^{1/5})$ electrical flow computations described in Theorem 2.2. These take place on (d), which has size about $O(n^{4/5})$. Once we have this approximate grouped flow, we can use the demands on the boundary vertices of each group to convert it back to a flow in that group in (b) that meets the same demand. Piecing these flows on all groups together gives a grouped flow in (b). The total iteration count of our algorithm is $\tilde{O}(r^{1/2}) \times \tilde{O}((n/r)^{1/3}) = \tilde{O}(n^{2/5})$. Although this is slightly higher than the Christiano et al. algorithm, our overall running time is faster due to the smaller size of (d). Specifically, since $\tilde{O}(n^{1/5})$ calls to (b) are made, the total running time becomes $\tilde{O}(n^{6/5})$.

4 Using Approximate Grouped Flows

Our algorithm uses approximate grouped flows in a way similar to the use of electrical flows in the Christiano et al. algorithm [CKM⁺11]. They showed that electrical flows can be used as an (ϵ, ρ) oracle as defined below.

Definition 4.1 For $\epsilon > 0$ and $\rho > 0$, an (ϵ, ρ) -flow oracle is an algorithm that given demand \mathbf{d} , a flow amount F and a vector $\mathbf{w}_{\text{oracle}}$ of edge weights with $\mathbf{w}_{\text{oracle}}(e) > 1$ for all e , returns one of the following:

1. If there exist a flow $\bar{\mathbf{f}}$ that routes F units of flow from s to t and has max congestion 1, then output a flow \mathbf{f} satisfying:
 - (a) \mathbf{f} routes F meets the demand, $\mathbf{B}^T \mathbf{f} = \mathbf{d}$
 - (b) $\sum_e \mathbf{w}_{\text{oracle}}(e) \text{cong}(\mathbf{f}, e) \leq (1 + \epsilon) |\mathbf{w}_{\text{oracle}}|_1$, where $|\mathbf{w}_{\text{oracle}}|_1 = \sum_e \mathbf{w}_{\text{oracle}}(e)$;
 - (c) $\text{cong}(\mathbf{f}, e) \leq \rho$ for all edges e .
2. Otherwise, it either outputs a flow \mathbf{f} satisfying conditions 1a, 1b, 1c or outputs “fail”.

Christiano et al. showed that obtaining (ϵ, ρ) -flow oracles is sufficient for approximate maximum flow algorithms [CKM⁺11]. We use Theorem 3.2 from their paper as our starting point:

Lemma 4.2 (Theorem 3.2 from [CKM⁺11]) For any $0 < \epsilon < 1/2$ and $\rho > 0$, given an (ϵ, ρ) -flow oracle with running time $T(m, 1/\epsilon)$, one can obtain an algorithm that computes a $(1 - O(\epsilon))$ -approximate maximum flow in a capacitated, undirected graph in time $\tilde{O}(\rho \epsilon^{-2} \cdot T(m, 1/\epsilon))$. Furthermore, all calls to the (ϵ, ρ) oracle have $U(\mathbf{w}) \leq O(m/\epsilon)$.

It was also shown Section 2.2 of [CKM⁺11] that for finding an approximate maximum flow, it suffices to consider capacities with $U(\mathbf{u}) \leq O(m/\epsilon)$. We show that grouped flows can be used to give (ϵ, ρ) -flow oracles. This is done by choosing the edge weights, \mathbf{w} based on the weights given to the oracle, $\mathbf{w}_{\text{oracle}}$. The following proof is directly based on the one in Section 3.2. of [CKM⁺11], which shows that electrical flows can be used as (ϵ, ρ) -flow oracles.

Proof of Lemma 3.2: By Lemma 4.2, it suffices to provide an $(\epsilon, O(r^{1/2}\epsilon^{-1/2}))$ -flow oracle to the Christiano et al. algorithm. Let graph that we’re given be $G = (V, E)$, with edge capacities \mathbf{u} ; the demand of the flow be \mathbf{d} and weights on edges be $\mathbf{w}_{\text{oracle}}$. Let $\mathbf{w}_{\text{oracle}}(G^{(i)})$ be the total weight among edges in component i , aka. $\mathbf{w}_{\text{oracle}}(G^{(i)}) = \sum_{e \in E(G^{(i)})} \mathbf{w}_{\text{oracle}}(e)$. We set the weight of an edge e in component i to:

$$\mathbf{w}(e) = \frac{1 - \epsilon/2}{\mathbf{u}(e)^2} \left(\frac{\mathbf{w}_{\text{oracle}}(e)}{\mathbf{w}_{\text{oracle}}(G^{(i)})} + \frac{\epsilon}{4|E(G^{(i)})|} \right)$$

Note that the assignments of edge weights have an extra $\mathbf{u}(e)^2$ term in the denominator due to $\mathbf{cong}(\mathbf{f}, e) = \frac{\mathbf{f}(e)}{\mathbf{u}(e)}$. This allows us to write the energy emitted on edge e in the following more convenient form:

$$\begin{aligned}\mathbf{w}(e)\mathbf{f}(e)^2 &= \frac{1 - \epsilon/2}{\mathbf{u}(e)^2} \left(\frac{\mathbf{w}_{oracle}(e)}{\mathbf{w}_{oracle}(G^{(i)})} + \frac{\epsilon}{4|E(G^{(i)})|} \right) \mathbf{f}(e)^2 \\ &= (1 - \epsilon/2) \left(\frac{\mathbf{w}_{oracle}(e)}{\mathbf{w}_{oracle}(G^{(i)})} + \frac{\epsilon}{4|E(G^{(i)})|} \right) \mathbf{cong}(\mathbf{f}, e)^2\end{aligned}$$

We first show that the existence of a flow $\bar{\mathbf{f}}$ where $\mathbf{cong}(\bar{\mathbf{f}}, e) \leq 1$ implies that $\mathbf{cong}(\bar{\mathbf{f}}, i) \leq 1 - \epsilon/10$:

$$\begin{aligned}\mathbf{cong}(\bar{\mathbf{f}}, i)^2 &= \sum_{e \in E(G^{(i)})} \mathbf{w}(e)\bar{\mathbf{f}}(e)^2 \\ &= (1 - \epsilon/2) \sum_{e \in E(G^{(i)})} \left(\frac{\mathbf{w}_{oracle}(e)}{\mathbf{w}_{oracle}(G^{(i)})} + \frac{\epsilon}{4|E(G^{(i)})|} \right) \mathbf{cong}(\bar{\mathbf{f}}, e)^2 \\ &\leq (1 - \epsilon/2) \sum_{e \in G^{(i)}} \left(\frac{\mathbf{w}_{oracle}(e)}{\mathbf{w}_{oracle}(G^{(i)})} + \frac{\epsilon}{4|E(G^{(i)})|} \right) 1 \\ &\leq (1 - \epsilon/2)(1 + \epsilon/4) \\ &\leq 1 - \epsilon/5\end{aligned}$$

Taking square roots of both sides gives the bound.

Thus if APPROXGROUPEDFLOW is run with an error bound of $\epsilon/10$, it returns a flow \mathbf{f} where $\mathbf{cong}(\mathbf{f}, i) \leq 1 + \epsilon/10$. This condition is equivalent to:

$$\begin{aligned}(1 + \epsilon/10)^2 &\geq \mathbf{cong}(\mathbf{f}, i)^2 \\ &= \sum_{e \in E(G^{(i)})} \mathbf{w}(e)\mathbf{f}(e)^2 \\ &\geq \mathbf{w}(e)\mathbf{f}(e)^2 \\ &= (1 - \epsilon/2) \sum_{e \in E(G^{(i)})} \left(\frac{\mathbf{w}_{oracle}(e)}{\mathbf{w}_{oracle}(G^{(i)})} + \frac{\epsilon}{4|E(G^{(i)})|} \right) \mathbf{cong}(\mathbf{f}, e)^2\end{aligned}$$

By the assumption that $\epsilon < 1/2$, we have $\frac{(1+\epsilon/10)^2}{1-\epsilon/2} \leq (1 + \epsilon)^2$. Taking this into account and rearranging gives:

$$(1 + \epsilon)^2 \geq \sum_{e \in E(G^{(i)})} \left(\frac{\mathbf{w}_{oracle}(e)}{\mathbf{w}_{oracle}(G^{(i)})} + \frac{\epsilon}{4|E(G^{(i)})|} \right) \mathbf{cong}(\mathbf{f}, e)^2$$

We now show that this flow meets the requirements of an $(\epsilon, O(r^{1/2}\epsilon^{-1/2}))$ -flow oracle, namely the following two conditions:

1. For each edge $e \in E(G^{(i)})$, $\mathbf{cong}(\mathbf{f}, e) \leq O(|E(G^{(i)})|^{1/2}\epsilon^{-1/2}) \leq O(r^{1/2}\epsilon^{-1/2})$
2. $\sum_{e \in E(G^{(i)})} \mathbf{w}_{oracle}(e)\mathbf{cong}(\mathbf{f}, e) \leq (1 + \epsilon)\mathbf{w}_{oracle}(G^{(i)})$.

For Part 1, we take the first term in the weights and isolate edge e to get:

$$(1 + \epsilon)^2 \geq \frac{\epsilon}{4|E(G^{(i)})|} \mathbf{cong}(\mathbf{f}, e)^2$$

Rearranging gives:

$$\begin{aligned} \mathbf{cong}(\mathbf{f}, e)^2 &\leq (1 + \epsilon)^2 4|E(G^{(i)})| \\ &\leq O(\epsilon^{-1}|E(G^{(i)})|) \end{aligned}$$

Taking square-roots of both sides again completes Part 1.

For Part 2, we take the second part of the weight terms to get:

$$(1 + \epsilon)^2 \geq \sum_{e \in E(G^{(i)})} \frac{\mathbf{w}_{oracle}(e)}{\mathbf{w}_{oracle}(G^{(i)})} \mathbf{cong}(\mathbf{f}, e)^2$$

Multiplying both sides again by $\mathbf{w}_{oracle}(G^{(i)})^2 = (\sum_{e \in E(G^{(i)})} \mathbf{w}_{oracle}(e))^2$ and applying the Cauchy-Schwarz inequality gives:

$$\begin{aligned} (1 + \epsilon)^2 \left(\sum_{e \in E(G^{(i)})} \mathbf{w}_{oracle}(e) \right)^2 &\geq \left(\sum_{e \in E(G^{(i)})} \mathbf{w}_{oracle}(e) \right) \left(\sum_{e \in E(G^{(i)})} \mathbf{w}_{oracle}(e) \mathbf{cong}(\mathbf{f}, e)^2 \right) \\ &\geq \left(\sum_{e \in E(G^{(i)})} \mathbf{w}_{oracle}(e) \mathbf{cong}(\mathbf{f}, e) \right)^2 \end{aligned}$$

Taking square roots of both sides gives Part 2.

It remains to bound the ratio between weights assigned to two edges e and e' , we have:

$$\begin{aligned} \mathbf{w}(e') &\leq \frac{1 - \epsilon/2}{\mathbf{u}(e')^2} (1 + \epsilon) \leq \frac{2}{\mathbf{u}(e')^2} \\ \mathbf{w}(e) &\geq \frac{1 - \epsilon/2}{\mathbf{u}(e)^2} \frac{\epsilon}{4|E(G^{(i)})|} \geq \frac{\epsilon}{4m\mathbf{u}(e)^2} \\ \frac{\mathbf{w}(e')}{\mathbf{w}(e)} &\leq \left(\frac{2}{\mathbf{u}(e')^2} \right) / \left(\frac{\mathbf{u}(e)^2 \epsilon}{4m} \right) \\ &\leq \frac{8m}{\epsilon} \left(\frac{\mathbf{u}(e)}{\mathbf{u}(e')} \right)^2 \leq O(m^3 \epsilon^{-3}) \end{aligned}$$

■

5 Converting Flows to and from Sparsifier

We now show that an approximate grouped flow can be obtained from an approximate grouped flow on the graph where each group is replaced with a spectral vertex-sparsifier. The differences between both vertices and edges of the two graphs makes a direct, embedding based mapping between flows on the edges difficult. As a result, our conversion process relies on computing locally optimal electrical flows and bounding their energy dissipations using guarantees of spectral sparsifiers. We only use the residue of the flows on the boundaries of each piece on to make this conversion. The energy of the optimum electrical flow given in Lemma 2.1 then allows us to (approximately) preserve the energy of flows between the original

graph and its spectral vertex sparsifier.

Recall that as goal is to find $s - t$ flows, we may assume that both s and t are on boundaries. More generally, we restrict ourselves to flows where internal vertices have zero demand. We first show that for the exact Schur complement, such flows can be converted exactly. The following lemma is an re-statement of results on bounding the support of Steiner tree based support tree preconditioners [Gre96, MMP⁺05, KM08]. The usual proof is given in Appendix C for completeness.

Lemma 5.1 *For any demand vector \mathbf{d} defined on the vertices of a graph partitioned into boundary and inner vertices such that the only non-zero demands are on the boundary vertices. Let \mathbf{d}_{bdry} be the restriction of \mathbf{d} onto the boundary vertices and \mathbf{L}_{schur} be $\text{SCHURCOMPLEMENT}(\mathbf{L}, V_{intr})$. Then we have:*

$$\mathbf{d}^T \mathbf{L}^+ \mathbf{d} = \mathbf{d}_{bdry}^T \mathbf{L}_{schur}^+ \mathbf{d}_{bdry}$$

By Lemma 2.1, these two terms correspond to energies of electrical flows on the two graphs. This shows that if all inner vertices have zero demand, the demands on the boundary determine the minimum energy. However, due to the need to work with smaller problem instances, we can only use an approximation of \mathbf{L}_{schur} . Let this approximate spectral vertex sparsifier be $\tilde{\mathbf{L}}_{schur}$. The approximation guarantee given by Definition 3.4 is $(1 - \epsilon)\mathbf{L}_{schur} \preceq \tilde{\mathbf{L}}_{schur} \preceq (1 + \epsilon)\mathbf{L}_{schur}$. This error of $1 \pm \epsilon$ can be accounted for in the energy difference of the flows. This leads to the following Lemma, which allows us to convert between flows on two graphs whose Schur complement are similar. Note that since the spectral vertex sparsifier is only on the boundary vertices, it's a Schur complement of itself.

Lemma 5.2 *Given a graph $G = (V(G), E(G), \mathbf{w}(G))$ and $H = (V(H), E(H), \mathbf{w}(H))$ be connected graphs, with $V_{bdry}(G) = V_{bdry}(H)$ and:*

$$\text{SCHURCOMPLEMENT}(G, V_{bdry}(G)) \preceq (1 + \epsilon) \text{SCHURCOMPLEMENT}(H, V_{bdry}(H))$$

If $\mathbf{f}(G)$ is a flow on G that meets demand $\mathbf{d}(G)$ that's only non-zero on boundary vertices. Then we can find $\mathbf{f}(H)$ on H such that the residuals of $\mathbf{f}(H)$ equals to \mathbf{d}_{bdry} on boundary vertices, and are 0 in all interior vertices, and:

$$\mathcal{E}_{\mathbf{f}(H)}(\mathbf{w}(H)) \leq (1 + 3\epsilon) \mathcal{E}_{\mathbf{f}(G)}(\mathbf{w}(G))$$

Furthermore, $\mathbf{f}(H)$ can be found in $\tilde{O}(|E(G)| + |E(H)| \log(U(\mathbf{w}(H))/\epsilon))$ time.

Proof Since G and H are connected, the null spaces of their Schur complements onto the boundary are the same. If we denote them as $\mathbf{L}_{schur}(G)$ and $\mathbf{L}_{schur}(H)$, the giving condition becomes:

$$\mathbf{L}_{schur}(H) \preceq (1 + \epsilon) \mathbf{L}_{schur}(G)$$

Let $\mathbf{d}(G)$ and $\mathbf{d}(H)$ be the vectors formed by extending \mathbf{d}_{bdry} on to $V(G)$ and $V(H)$. Substituting \mathbf{d}_{bdry} into this bound and applying Lemma 5.1 gives:

$$\begin{aligned} \mathbf{d}(H)^T \mathbf{L}(H)^+ \mathbf{d}(H) &= \mathbf{d}_{bdry}^T \mathbf{L}_{schur}(H)^+ \mathbf{d}_{bdry} \\ &\leq (1 + \epsilon) \mathbf{d}_{bdry}^T \mathbf{L}_{schur}(G)^+ \mathbf{d}_{bdry} \\ &= (1 + \epsilon) \mathbf{d}(G)^T \mathbf{L}(G)^+ \mathbf{d}(G) \end{aligned}$$

By Lemma 2.1, $\mathcal{E}_{\mathbf{f}(G)}(\mathbf{w}(G)) \geq \mathbf{d}(G)^T \mathbf{L}(G)^+ \mathbf{d}(G)$. This in turns implies that minimum energy flow that routes $\mathbf{d}(H)$ in H has energy at most $(1 + \epsilon) \mathcal{E}_{\mathbf{f}(G)}(\mathbf{w}(G))$. Invoking Theorem 2.2 with $\delta = \epsilon$ then

returns a flow whose energy is at most:

$$\begin{aligned} (1 + \epsilon)\mathcal{E}(\mathbf{w}(H)) &\leq (1 + \epsilon)^2\mathcal{E}(\mathbf{w}(G)) \\ &\leq (1 + 3\epsilon)\mathcal{E}_{\mathbf{f}(G)}(\mathbf{w}(G)) \end{aligned}$$

■

Note that since $\mathbf{d}(G) = \mathbf{B}(G)\mathbf{f}(G)$ and each edge belongs to one of the groups, it can be easily decomposed into a set of k demands, one per group. We can find the residue from all edges in each group, and apply this conversion lemma to them. This leads to an algorithm for converting flows between graphs where each group are spectrally equivalent. Its pseudocode is shown in Algorithm 1.

Algorithm 1 Algorithm for Converting flow between two graphs where Schur complements of all groups are close

APPROXGROUPEDFLOW

Input: Graphs G and H , along with partitions of edges into groups $G^{(1)}, \dots, G^{(k)}$ and $H^{(1)}, \dots, H^{(k)}$ such that $V_{bdry}(G^{(i)}) = V_{bdry}(H^{(i)})$ and $\mathbf{L}_{schur}(H^{(i)}) \preceq (1 + \epsilon)\mathbf{L}_{schur}(G^{(i)})$. Flow on $\mathbf{f}(G)$ meeting demand $\mathbf{d}(G)$ such that $\mathbf{d}(G)$ is only non-zero on boundary vertices. Error bound ϵ .

Output: Flow on H , $\mathbf{f}(H)$ meeting $\mathbf{d}(H)$ such that $\mathbf{d}_{bdry}(H) = \mathbf{d}_{bdry}(G)$, $\mathbf{d}(H)$ is 0 on all interior vertices, and $\text{cong}(\mathbf{f}(G), i) \leq (1 + 3\epsilon)\text{cong}(\mathbf{f}(H), i)$.

- 1: Initialize $\mathbf{f}(H) = 0$
 - 2: **for** $i = 1 \dots k$ **do**
 - 3: Compute the residuals of $\mathbf{f}(G)$ among edges in $E(G^{(i)})$, $\mathbf{d}^{(i)}(G)$
 - 4: Form $\mathbf{d}^{(i)}(H)$ by taking $\mathbf{d}_{bdry}^{(i)}(G)$ and putting 0 on all interior vertices
 - 5: $\mathbf{f}^{(i)}(H) \leftarrow \text{ELECTRICALFLOW}(G^{(i)}, \mathbf{d}^{(i)}, \mathbf{w}(i))$
 - 6: **end for**
 - 7: Sum $\mathbf{f}^{(1)}(H) \dots \mathbf{f}^{(k)}(H)$ to form $\mathbf{f}(H)$
 - 8: **return** $\mathbf{f}(H)$
-

Lemma 5.3 *Let G and H be graphs with edge partitions $G^{(1)}, \dots, G^{(k)}$ and $H^{(1)}, \dots, H^{(k)}$ such that $V_{bdry}G^{(i)} = V_{bdry}H^{(i)}$ and $\mathbf{L}_{schur}(H^{(i)}) \preceq (1 + \epsilon)\mathbf{L}_{schur}(G^{(i)})$. Given a flow $\mathbf{f}(G)$ meeting demand $\mathbf{d}(G)$ that's only non-zero on boundary vertices and error bound ϵ , $\text{CONVERT}(G, H, \mathbf{f}(G), \epsilon)$ produces in $\tilde{O}(|E(G)| + |E(H)| \log(U(\mathbf{w}(H))/\epsilon))$ time a flow $\mathbf{f}(H)$ such that:*

- $\mathbf{f}(H)$ meeting $\mathbf{d}(H)$ such that $\mathbf{d}_{bdry}(H) = \mathbf{d}_{bdry}(G)$, $\mathbf{d}(H)$ is 0 on all interior vertices.
- $\text{cong}(\mathbf{f}(G), i) \leq (1 + 3\epsilon)\text{cong}(\mathbf{f}(H), i)$.

Proof By the given spectral condition between $G^{(i)}$ and $H^{(i)}$ and Lemma 5.2, we have that:

$$\begin{aligned} \text{cong}(\mathbf{f}(H), i)^2 &= \mathcal{E}_{\mathbf{f}^{(i)}(H)}(\mathbf{w}(H^{(i)})) \\ &\leq (1 + 3\epsilon)\mathcal{E}_{\mathbf{f}^{(i)}(G)}(\mathbf{w}(G^{(i)})) \\ &= \text{cong}(\mathbf{f}(G), i)^2 \end{aligned}$$

Taking square roots of both sides gives the bound on congestion.

Note that since the computation of residuals is linear and each edge belongs to one partition, we have:

$$\mathbf{d}(G) = \sum_{i=1}^k \mathbf{d}^{(i)}(G)$$

$$\mathbf{d}(H) = \sum_{i=1}^k \mathbf{d}^{(i)}(H)$$

Since $\mathbf{d}_{bdry}^{(i)}(G) = \mathbf{d}_{bdry}^{(i)}(H)$ and $\mathbf{d}^{(i)}(H)$ are 0 on interior vertices, we satisfy the requirement on $\mathbf{d}(H)$. ■

The faster algorithm that takes advantage of the ϵ -spectral vertex sparsifiers is now clear. We take the restriction of \mathbf{d} onto the sparsified graph, and find an approximate grouped L_2 flow on it. Since each group now only consists of their boundary vertices, they're smaller by a factor of $\tilde{O}(r^{1/2})$, which compensates for the $\tilde{O}(k^{1/3})$ iteration count from Theorem 3.3. Once an approximate grouped flow is found on the smaller graph, we convert it to one that approximately meets the constraints on the original, unsparified graph. Pseudocode of this algorithm is shown in Algorithm 2.

Algorithm 2 Faster Approximate Grouped L_2 Flows Using r -divisions

APPROXGROUPEDFLOW

Input: Weighted graph $G = (V, E, \mathbf{w})$ with an r -division partitioning it into groups $G^{(1)}, \dots, G^{(k)}$. Demand vector \mathbf{d} such that only vertices on boundary of groups can have non-zero demand. Error bound ϵ . $\epsilon/10$ spectral vertex sparsifiers $\tilde{\mathbf{L}}_{schur}(G^{(i)})$ for all the groups.

Output: Approximate grouped L_2 flow on G with congestion $1 + \epsilon$

- 1: Construct \tilde{G}_{schur} by combining $\tilde{\mathbf{L}}_{schur}(G^{(i)})$ for $1 \leq i \leq k$.
 - 2: Let \mathbf{d}_{schur} be the restriction of \mathbf{d} onto the $V(\tilde{G}_{schur})$
 - 3: $\tilde{\mathbf{f}} \leftarrow \text{GROUPEDFLOW}(\tilde{G}_{schur}, E(G^{(1)}) \dots E(G^{(k)}), \mathbf{d}, \epsilon/2)$
 - 4: Let \mathbf{f} be $\text{CONVERT}(G, \tilde{G}_{schur}, \tilde{\mathbf{f}}, \epsilon/10)$
 - 5: **return** \mathbf{f}
-

Proof of Theorem 3.5: We first show that the flow produced meets the requirement. By the requirement of spectral vertex sparsifiers given in Definition 3.4, we have:

$$(1 - \epsilon/10)\mathbf{L}_{schur}(G^{(i)}) \preceq_{\tilde{G}_{schur}} \tilde{G}_{schur}^{(i)} \preceq (1 + \epsilon/10)\mathbf{L}_{schur}(G^{(i)})$$

Assume there is a grouped L_2 flow in $G, \bar{\mathbf{f}}$ with congestion at most $1 - \epsilon$. Then applying Lemma 5.3 with error $\epsilon/10$ shows the existence a grouped L_2 flow in \tilde{G}_{schur} with congestion at most $(1 - \epsilon)(1 + 3/10\epsilon) \leq 1 - \epsilon/2$. Therefore by the guarantees of the grouped flow algorithm given in Theorem 3.3, we get that $\tilde{\mathbf{f}}$ has congestion at most $1 + \epsilon/2$. Applying Lemma 5.3 in the reverse direction, this time algorithmically, gives a grouped L_2 flow in G with congestion at most $1 + \epsilon$.

It remains to bound the running time, which consists of running GROUPEDFLOW on the vertex-sparsified graph, and converting the flow back. The cost of conversion is bounded by $\tilde{O}(m \log(U\epsilon^{-1}))$ and the cost of grouped L_2 flow follows by the total number of edges in the components and $k = O(m/r)$. ■

6 Comments / Future Work

For readers who are familiar with the Christiano et al. algorithm, an alternate view of our algorithm that's closer to the multiplicative weights update framework is that it takes advantages of the grouping by making more aggressive readjustments of all edges in the same group. In this view we track and adjust another

set of weights defined for each component which are adjusted by the grouped L_2 flow algorithm, while the per edge weights are adjusted less frequently in an outer layer. In the electrical flow computations, the resistive value of an edge then depends on both its weight and that of its component.

To better understand the costs of our algorithm it's helpful to give a break down of the costs incurred by each component, ignoring constants and logarithmic terms. Such a list is given in Figure 2, where cost is the total cost incurred by all calls made by the component and number of calls is the total number of times this component is called in order to compute an approximate maximum flow. We now go over these procedures, their costs, and calling order. To compute a maximum flow we average $O(r^{1/2})$, the width of the oracle, grouped flows on the full graph. The grouped flow procedure makes several calls. It first computes spectral vertex sparsifier on each of the (n/r) groups. Then it computes a grouped flow on the sparsified graph by calling electrical flow $(n/r)^{1/3}$ times. The grouped L_2 flow on the sparsified graph is then converted to the full graph group-by-group.

Procedures	Size	Number of Calls	Cost	Total Cost
Grouped L_2 Flow on full graph	n	$r^{\frac{1}{2}}$	n	$nr^{\frac{1}{2}}$
Spectral Vertex Sparsification	r	$r^{\frac{1}{2}} \times n/r$	r	$nr^{\frac{1}{2}}$
Grouped L_2 Flow on Sparsifier	$nr^{-\frac{1}{2}}$	$r^{\frac{1}{2}}$	n	$nr^{\frac{1}{2}}$
Electrical Flow on Sparsifier	$nr^{-\frac{1}{2}}$	$r^{\frac{1}{2}} \times (n/r)^{\frac{1}{3}}$	$nr^{-\frac{1}{2}}$	$n^{\frac{4}{3}}r^{-\frac{1}{3}}$
Extrapolation of grouped L_2 flow to full graph	r	$r^{\frac{1}{2}} \times \frac{n}{r}$	r	$nr^{\frac{1}{2}}$

Figure 2: Summary of components and their costs in a sparse graph parameterized in terms of r , the size of groups in the r -division

Our recursive approach extends readily to multiple levels, but does not lead to better running times when we make more gradual reductions in graph sizes. The main reason is that the computation of spectral vertex sparsifiers in Appendix B first computes a dense and almost exact vertex sparsifier. The size of this dense graph at $O(|V_{bdry}(G^{(i)})|^2)$ is the bottleneck, although our algorithm only uses a sparsified version of it. We believe constructing the final spectral vertex sparsifier directly from the original graph is an interesting question on its own. On the other hand, for graphs with better separator structures such as graphs with bounded or low tree-width, this term is much smaller and improved running times are likely.

Another possible way to obtain a speedup is to reduce the width of grouped flows to $O(r^{1/3} \text{poly}(1/\epsilon))$ in a way similar to the Christiano et al. algorithm [CKM⁺11]. The difficulty here is that their analysis requires tracking terms up to precision of roughly $r^{-1/3}$, but going through spectral vertex sparsifiers incurs an error of ϵ already. As a result, improvements in this direction will likely require a more robust analysis of the Christiano et al. algorithm.

Finally, it's worth noting that all known hard instances for the current analysis of the Christiano et al. algorithm, such as the one given in Section 4 of [CKM⁺11] have good separator structures. Whether this type of divide and conquer approach can lead to speed-ups for approximate maximum flow on general graphs is an intriguing direction for future work.

Acknowledgements

We thank the anonymous reviewers for their very helpful comments.

References

- [BCM08] A. Buades, B. Coll, and J.M Morel. Nonlocal image and movie denoising. *IJCV*, 76(2):123–139, 2008. 1
- [BK09] Glencora Borradaile and Philip N. Klein. An $o(n \log n)$ algorithm for maximum st -flow in a directed planar graph. *J. ACM*, 56(2), 2009. 1

-
- [BKM⁺11] Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. In *FOCS*, pages 170–179, 2011. [1](#)
 - [Bor08] Glencora Borradaile. *Exploiting planarity for network flow and connectivity problems*. PhD thesis, Brown University, Providence, RI, USA, 2008. AAI3318294. [1](#)
 - [CEN09a] Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Homology flows, cohomology cuts. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 273–282, New York, NY, USA, 2009. ACM. [1](#)
 - [CEN09b] Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Minimum cuts and shortest homologous cycles. In *Proceedings of the 25th annual symposium on Computational geometry*, SCG '09, pages 377–385, New York, NY, USA, 2009. ACM. [1](#)
 - [CKM⁺11] Paul Christiano, Jonathan A. Kelner, Aleksander Mądry, Daniel Spielman, and Shang-Hua Teng. Electrical Flows, Laplacian Systems, and Faster Approximation of Maximum Flow in Undirected Graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, 2011. [1](#), [2.3](#), [2.3](#), [3](#), [3](#), [4](#), [4](#), [4.2](#), [4](#), [6](#), [D](#)
 - [DS84] Peter G. Doyle and J. Laurie Snell. *Random Walks and Electric Networks*, volume 22 of *Carus Mathematical Monographs*. Mathematical Association of America, 1984. [2.3](#), [B](#)
 - [FF56] L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. [1](#)
 - [Fre87] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987. [2.4](#)
 - [Goo95] Michael T. Goodrich. Planar separators and parallel polygon triangulation. *J. Comput. Syst. Sci.*, 51(3):374–389, 1995. [2.4](#)
 - [GR98] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45:783–797, September 1998. [1](#)
 - [Gre96] Keith Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*. PhD thesis, Carnegie Mellon University, Pittsburgh, October 1996. CMU CS Tech Report CMU-CS-96-123. [5](#)
 - [GT87] J. R. Gilbert and R. E. Tarjan. The analysis of a nested dissection algorithm. *Numerische Mathematik*, 50(4):377–404, 1987. [2.4](#)
 - [GY04] Donald Goldfarb and Wotao Yin. Second-order cone programming methods for total variation-based image restoration. *SIAM J. Sci. Comput.*, 27:622–645, 2004. [1](#)
 - [Has81] Refael Hassin. Maximum flow in (s, t) planar networks. *Inf. Process. Lett.*, 13(3):107, 1981. [1](#)
 - [HMMP12] Hui Han Chin, Aleksander Mądry, Gary L. Miller, and Richard Peng. Runtime guarantees for regression problems. *CoRR*, abs/1110.1358, 2012. [1](#), [3](#), [3](#), [A](#)
 - [HR56] T. E. Harris and F. S. Ross. Fundamentals of a method for evaluating rail net capacities. Technical Report RM-1573, 1956. [1](#)
 - [II90] Hiroshi Imai and Kazuo Iwano. Efficient sequential and parallel algorithms for planar minimum cost flow. In *Proceedings of the International Symposium on Algorithms*, SIGAL '90, pages 21–30, London, UK, UK, 1990. Springer-Verlag. [1](#)
 - [INSWN11] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 313–322, New York, NY, USA, 2011. ACM. [1](#)
 - [JV82] D.B. Johnson and S.M. Venkatesan. Using divide and conquer to find flows in directed planar networks in $O(n^{3/2} \log n)$ time. In *20th annual allerton conference on communication, control and computing*, pages 898–905, 1982. [1](#)
 - [Kar98] David R. Karger. Better random sampling algorithms for flows in undirected graphs. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, SODA '98, pages 490–499, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics. [1](#)

-
- [KL11] Jonathan A. Kelner and Alex Levin. Spectral Sparsification in the Semi-Streaming Setting. In Thomas Schwentick and Christoph Dürr, editors, *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 440–451, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. 3, B.9
 - [KM08] Ioannis Koutis and Gary L. Miller. Graph partitioning into isolated, high conductance clusters: Theory, computation and applications to preconditioning. In *Symposium on Parallel Algorithms and Architectures (SPAA)*, 2008. 5
 - [KMP10a] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD systems. *CoRR*, abs/1003.2958, 2010. 1
 - [KMP10b] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD systems. In *FOCS '10: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, 2010. B.4
 - [KMP11a] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly $m \log n$ solver for SDD linear systems. In *FOCS '11: Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, 2011. B.4
 - [KMP11b] Ioannis Koutis, Gary L. Miller, and Richard Peng. Solving sdd linear systems in time $\tilde{O}(m \log n \log(1/\epsilon))$. *CoRR*, abs/1102.4842, 2011. 1
 - [KR10] K. Kawarabayashi and B. Reed. A separator theorem in minor-closed classes. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 153–162, oct. 2010. 1, 1, 2.4
 - [LF10] Ce Liu and William T. Freeman. A high-quality video denoising algorithm based on reliable motion estimation. In *Proceedings of the 11th European conference on computer vision conference on Computer vision: Part III, ECCV'10*, pages 706–719, Berlin, Heidelberg, 2010. Springer-Verlag. 1
 - [LRT79] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM J. on Numerical Analysis*, 16:346–358, 1979. 1, 2.4, 2.4, B
 - [LT79] R. J. Lipton and R. E. Tarjan. A planar separator theorem. *SIAM J. Appl. Math.*, 36:177–189, 1979. 1, 1, 2.4
 - [M10] Aleksander Mądry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 121–130, New York, NY, USA, 2010. ACM. 1
 - [MMP⁺05] Bruce M. Maggs, Gary L. Miller, Ojas Parekh, R. Ravi, and Shan Leung Maverick Woo. Finding effective support-tree preconditioners. In *Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, SPAA '05, pages 176–185, New York, NY, USA, 2005. ACM. 5
 - [MN95] Gary L. Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM Journal on Computing*, 24:1002–1017, 1995. 1, 1
 - [MTTV97] Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Separators for sphere-packings and nearest neighborhood graphs. *JACM*, 44(1):1–29, January 1997. 1, 2.4
 - [PR93] Victor Pan and John Reif. Fast and efficient parallel solution of sparse linear systems. *SIAM J. Comput.*, 22(6):1227–1250, December 1993. 3
 - [SS08] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 563–568, 2008. 3, 3, 3, B, B.9
 - [ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 81–90, June 2004. 1, B.4
 - [ST06] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105, 2006. 1, B.4
 - [VSN09] Mayank Vatsa, Richa Singh, and Afzel Noore. Denoising and segmentation of 3D brain images. In *Proceedings of the 2009 International Conference on Image Processing, Computer Vision, & Pattern Recognition, IPCV 2009, July 13-16, 2009, Las Vegas, Nevada, USA, 2 Volumes*, pages 561–567. CSREA Press, 2009. 1
 - [WN11] Christian Wulff-Nilsen. Separator theorems for minor-free and shallow minor-free graphs with applications. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 37–46, Washington, DC, USA, 2011. IEEE Computer Society. 1, 1, 2.4, 3

A Algorithm for Approximating Grouped L_2 Flow

We now show that given an undirected flow instance where the edges are grouped into k groups, the maximum L_2 energy among the groups can be approximately minimized using about $\tilde{O}(k^{1/3}\epsilon^{-8/3})$ iterations. The algorithm assigns weights to all the groups and adjusts them iteratively. We will use $^{(t)}$ to denote that the variable is associated with the t -th iteration. Pseudocode of the algorithm is shown in Algorithm 3.

Algorithm 3 Algorithm for grouped flows

GROUPEDFLOW

Input: Weighted, undirected graph $G = (V, E, S_1, \dots, S_k)$ with edges partitioned into $S_1 \dots S_k$ and weights on edges given by $\mathbf{w} : E \rightarrow \mathbb{R}^+$. Demand vector \mathbf{d} and error bound ϵ .

Output: Either a flow \mathbf{f} that meets the demands given by \mathbf{d} and that $\text{cong}(\mathbf{f}, i) \leq 1 + 10\epsilon$. Or “fail” indicating that no such flow with maximum congestion 1 exists.

```

1:  $\rho \leftarrow 10k^{1/3}\epsilon^{-2/3}$ 
2:  $N \leftarrow 20\rho \ln(k)\epsilon^{-2} = 200k^{1/3} \ln(k)\epsilon^{-8/3}$ 
3: Initialize  $\mathbf{w}_{grp}^{(0)}(i) = 1$  for all groups  $1 \leq i \leq k$ 
4:  $\mathbf{f} \leftarrow \mathbf{0}$ ,  $N_1 \leftarrow 0$ 
5: for  $t = 1 \dots N$  do
6:   Compute  $\mu^{(t-1)} \leftarrow \sum_i \mathbf{w}_{grp}^{(t-1)}(i)$ 
7:   Set  $\mathbf{r}(e)$  to  $(\mathbf{w}_{grp}^{(t-1)}(i) + \frac{\epsilon}{k}\mu^{(t-1)}) \cdot \mathbf{w}(e)$  for all edges  $e$  in group  $S_i$ 
8:    $\tilde{\mathbf{f}}^{(t)} = \text{ELECTRICALFLOW}(G = (V, E), \mathbf{d}, \mathbf{r})$ 
9:   if  $\mathcal{E}_{\tilde{\mathbf{f}}^{(t)}}(\mathbf{r}) > \mu^{(t-1)}$  then
10:    return “fail”
11:   else
12:     if  $\text{cong}(\mathbf{f}, i) \leq \rho$  for all  $i$  then
13:        $\mathbf{f} \leftarrow \mathbf{f} + \tilde{\mathbf{f}}^{(t)}$ 
14:        $N_1 \leftarrow N_1 + 1$ 
15:     end if
16:     Update weights for all groups,  $\mathbf{w}_{grp}^{(t)}(i) \leftarrow \mathbf{w}_{grp}^{(t-1)}(i) \left(1 + \frac{\epsilon}{\rho} \text{cong}(\mathbf{f}, i)\right)$ 
17:   end if
18: end for
19: return  $\frac{1}{N_1} \mathbf{f}$ 

```

We first state the following bounds regarding the overall sum of potentials $\mu^{(t)}$, the weight of a single group $\mathbf{w}^{(t)}(i)$ and the effective conductance given by the reweighed energy matrices at each iteration, $\mathcal{E}^{(t)}$.

Lemma A.1 *The following holds when $\tilde{\mathbf{f}}^{(t)}$ satisfies*

$$\sum_i \mathbf{w}_{grp}^{(t)}(i) \text{cong}(\tilde{\mathbf{f}}^{(t)}, i) \leq \mu^{(t-1)}$$

$$1. \mu^{(t)} \leq \exp\left(\frac{\epsilon}{\rho}\right) \mu^{(t-1)}$$

2. $w_{grp}^{(t)}(i)$ is non-decreasing across iterations t , and if $\text{cong}(\tilde{\mathbf{f}}^{(t)}, i) \leq \rho$ we have:

$$w_{grp}^{(t)}(i) \geq \exp\left(\frac{\epsilon}{\rho} \text{cong}(\tilde{\mathbf{f}}^{(t)}, i)\right) w_{grp}^{(t-1)}(i)$$

3. If for some edge e we have $\text{cong}(\tilde{\mathbf{f}}^{(t)}, i) \geq \rho$, then $\mathcal{E}(\mathbf{r}^{(t)}) \geq \mathcal{E}(\mathbf{r}^{(t-1)}) \exp\left(\frac{\epsilon^2 \rho^2}{4k}\right)$

The proof of Lemma A.1 relies on the following facts about $\exp(x)$ when x is close to 1:

Fact A.2 1. If $x \geq 0$, $1 + x \leq \exp(x)$.

2. If $0 \leq x \leq \epsilon$, then $1 + x \geq \exp((1 - \epsilon)x)$.

Proof of Part 1:

$$\begin{aligned} \mu^{(t)} &= \sum_i w_{grp}^{(t)}(i) \\ &= \sum_i w_{grp}^{(t-1)}(i) \left(1 + \frac{\epsilon}{\rho} \text{cong}(\tilde{\mathbf{f}}^{(t)}, i)\right) && \text{By the update rule} \\ &= \left(\sum_i w_{grp}^{(t-1)}(i)\right) + \frac{\epsilon}{\rho} \sum_i w_{grp}^{(t-1)}(i) \text{cong}(\tilde{\mathbf{f}}^{(t)}, i) \\ &= \mu^{(t-1)} + \frac{\epsilon}{\rho} \sum_i w_{grp}^{(t-1)}(i) \text{cong}(\tilde{\mathbf{f}}^{(t)}, i) && \text{By definition of } \mu^{(t-1)} \\ &\leq \mu^{(t-1)} + \frac{\epsilon}{\rho} \mu^{(t-1)} && \text{By bound on total weighted congestion} \\ &= \left(1 + \frac{\epsilon}{\rho}\right) \mu^{(t-1)} \\ &\leq \exp\left(\frac{\epsilon}{\rho}\right) \mu^{(t-1)} && \text{By Fact A.2.1} \end{aligned}$$

■(Part 1)

Proof of Part 2:

If $\text{cong}(\tilde{\mathbf{f}}^{(t)}, i) \leq \rho$, then $\frac{\epsilon}{\rho} \text{cong}(\tilde{\mathbf{f}}^{(t)}, i) \leq \epsilon$ and:

$$\begin{aligned} w_{grp}^{(t)}(i) &= w_{grp}^{(t-1)}(i) \left(1 + \frac{\epsilon}{\rho} \text{cong}(\tilde{\mathbf{f}}^{(t)}, i)\right) \\ &\leq w_{grp}^{(t-1)}(i) \exp\left(\frac{\epsilon(1 - \epsilon)}{\rho} \text{cong}(\tilde{\mathbf{f}}, i)\right) && \text{By Fact A.2.2} \end{aligned}$$

■(Part 2)

In order to prove Part 3, we need the following lemma was proven in [HMMP12] about increasing the weight of a heavily congested edge.

Lemma A.3 Let $\bar{\mathbf{f}}$ be the optimal electrical flow with flow value F on a graph G with resistances \mathbf{r} . Suppose there is a subset of the edges $S \subseteq E$ that accounts for β of the total energy of $\bar{\mathbf{f}}$, i.e.

$$\sum_{e \in S} \bar{\mathbf{f}}(e)^2 \mathbf{r}(e) \geq \beta \mathcal{E}(\mathbf{r}, \bar{\mathbf{f}})$$

For some $\gamma > 0$, define new resistances \mathbf{r}' such that $\mathbf{r}'(e) = (1 + \epsilon)\mathbf{r}(e)$ for all $e \in S$ and $\mathbf{r}'(e) = \mathbf{r}(e)$ for all $e \notin S$, then

$$\mathcal{E}(\mathbf{r}') \leq \exp(-\frac{\epsilon\beta}{2})\mathcal{E}(\mathbf{r})$$

Proof of Part 3:

Let i be the group such that $\mathbf{cong}(\tilde{\mathbf{f}}, i) \geq \rho$, then since $\mathbf{w}_{grp}(i) \geq \frac{\epsilon}{k}\mu$:

$$\begin{aligned} \mathbf{w}_{grp}^{(t-1)}(i)\mathbf{cong}(\tilde{\mathbf{f}}^{(t)}, i)^2 &\geq \frac{\epsilon}{k}\mu^{(t-1)}\rho^2 \\ &\geq \frac{\epsilon\rho^2}{k}\mathcal{E}(\mathbf{r}^{(t-1)}, \tilde{\mathbf{f}}) \quad \text{By assumption of the energy of the flow returned} \end{aligned}$$

Invoking the guarantees given in proven in Part 3 of Theorem 2.2, we have:

$$\begin{aligned} \sum_{e \in S_i} \mathbf{r}^{(t)}(e)\tilde{\mathbf{f}}(e)^2 &\geq \sum_{e \in S_i} \mathbf{r}^{(t-1)}(e)\mathbf{cong}(\tilde{\mathbf{f}}, e)^2 - \left| \mathbf{r}^{(t-1)}(e)\mathbf{cong}(\tilde{\mathbf{f}}, e)^2 - \mathbf{r}^{(t-1)}(e)\mathbf{cong}(\tilde{\mathbf{f}}, e)^2 \right| \\ &\geq \mathbf{cong}(\tilde{\mathbf{f}}^{(t)}, i) - \delta\mathcal{E}(\mathbf{r}^{(t-1)}) \quad \text{By Part 3 of Theorem 2.2} \\ &\geq \frac{\epsilon\rho^2}{k}\mathcal{E}_{\tilde{\mathbf{f}}^{(t)}}(\mathbf{r}^{(t-1)}) - \delta\mathcal{E}(\mathbf{r}^{(t-1)}) \quad \text{By Equation 3.1} \\ &\geq \frac{\epsilon\rho^2}{(1+\delta)k}\mathcal{E}(\mathbf{r}^{(t-1)}) - \delta\mathcal{E}(\mathbf{r}^{(t-1)}) \quad \text{By Part 2} \\ &\geq \frac{\epsilon\rho^2}{2k}\mathcal{E}(\mathbf{r}^{(t-1)}) \end{aligned}$$

Applying Lemma A.3 with $\beta = \frac{\epsilon\rho^2}{4k}$ completes the proof.

■(Part 3)

Proof of Theorem 3.3:

Since $\sum_{1 \leq i \leq k} (\mathbf{w}_{grp}^{(t-1)}(i) + \frac{\epsilon}{k}\mu^{(t-1)}) = (1 + \epsilon)\mu^{(t-1)}$, if there exist a flow \mathbf{f} such that $\mathbf{cong}(\mathbf{f}, i) \leq 1 - 2\epsilon$ for all e , we have that $\mathcal{E}(\mathbf{r}^{(t)}) \leq (1 - \epsilon)\mu^{(t-1)}$. Then if the algorithm does not return “fail”, Theorem 2.2 means that $\tilde{\mathbf{f}}^{(t)}$ satisfies:

$$\begin{aligned} \mathcal{E}_{\tilde{\mathbf{f}}^{(t)}}(\mathbf{r}^{(t)}) &\leq (1 + \delta)(1 - \epsilon)\mu^{(t-1)} \\ &\leq \mu^{(t-1)} \\ \sum_i \mathbf{w}_{grp}^{(t-1)}(i)\mathbf{cong}(\tilde{\mathbf{f}}^{(t)}, i)^2 &\leq \sum_i \mathbf{w}_{grp}^{(t-1)}(i) \end{aligned}$$

Multiplying both sides by $\mu^{(t-1)}$ and applying the Cauchy-Schwarz inequality gives:

$$\begin{aligned} \left(\sum_i \mathbf{w}_{grp}^{(t-1)}(i) \right)^2 &\geq \left(\sum_i \mathbf{w}_{grp}^{(t-1)}(i) \right) + \left(\sum_i \mathbf{w}_{grp}^{(t-1)}(i)\mathbf{cong}(\tilde{\mathbf{f}}^{(t)}, i)^2 \right) \\ &\geq \left(\sum_i \mathbf{w}_{grp}^{(t-1)}(i)\mathbf{cong}(\tilde{\mathbf{f}}^{(t)}, i) \right) \end{aligned}$$

Taking the square root of both sides gives:

$$\sum_i \mathbf{w}_{grp}^{(t-1)}(i) \mathbf{cong}(\tilde{\mathbf{f}}^{(t)}, i) \leq \mu^{(t-1)}$$

Therefore inductively applying Lemma A.1 Part 1, we have:

$$\begin{aligned} \mu^{(N)} &\leq \mu^{(0)} \cdot \left(\exp\left(\frac{\epsilon}{\rho}\right) \right)^N \\ &= \exp\left(\frac{\epsilon N}{\rho}\right) m \\ &\leq \exp\left(\frac{21 \ln m}{\epsilon}\right) \end{aligned}$$

We now bound N' , the number of iterations t where there is a group with $\mathbf{cong}(\tilde{\mathbf{f}}^{(t)}, i) \geq \rho$. Suppose $\mathcal{E}(\mathbf{r}^{(0)}) \leq 1/2$, then in the flow returned, no group has $\mathbf{cong}(\tilde{\mathbf{f}}^{(0)}, i) \geq 1$, which means that the algorithm can already return that flow.

Note that $\mathbf{r}^{(t)}$ is monotonically increasing on each edge, so $\mathcal{E}(\mathbf{r}^{(t)})$ is also monotonic. Then by Lemma A.1 Part 3, we have:

$$\mathcal{E}(\mathbf{r}^{(N)}) \geq 1/2 \cdot \exp\left(\frac{\epsilon^2 \rho^2 N'}{4m}\right)$$

Combining this with $\mathcal{E}(\mathbf{r}^{(t)}) \leq \mu^{(N)}$ gives:

$$\begin{aligned} \frac{\epsilon^2 \rho^2 N'}{4m} &\leq \frac{21 \ln m}{\epsilon} \\ N' &\leq \frac{100m \ln m}{\rho^2 \epsilon^3} \leq \epsilon N \end{aligned}$$

Then in all the $N - N' \geq (1 - \epsilon)N$ iterations, we have $\mathbf{cong}(\tilde{\mathbf{f}}^{(t)}, i) \leq \rho$ for all groups i . This allows us to finish our proof by bounding the congestion of each group:

$$\begin{aligned} \mathbf{cong}\left(\sum_t \tilde{\mathbf{f}}^{(t)}, i\right) &\leq \sum_t \mathbf{cong}(\tilde{\mathbf{f}}^{(t)}, i) \quad \text{Since } \mathbf{cong}(\cdot, i) \text{ is a } L_2 \text{ norm} \\ &\leq \log(\mu^{(t)}) / \left(\frac{\epsilon}{\rho}\right) \quad \text{By Lemma A.1 Part 2} \\ &= \frac{1}{1 - \epsilon} T' \leq (1 + 2\epsilon) T' \end{aligned}$$

■

B Spectral Vertex Sparsification

This section described faster algorithms for spectral vertex sparsification. As the Schur complements of the Laplacian of a disconnected graph is the sum of the Schur complement of its components' graph Laplacians, we assume that the graphs given as input are connected. We will use the following lemma about the Schur complements of spectrally similar PSD matrices.

Lemma B.1 Suppose $\mathbf{L}(G)$ and $\mathbf{L}(H)$ are matrices such that:

$$(1 - \epsilon)\mathbf{L}(G) \preceq \mathbf{L}(H) \preceq (1 + \epsilon)\mathbf{L}(G)$$

Then let $\mathbf{L}_{schur}(G)$ and $\mathbf{L}_{schur}(H)$ be the Schur complements of $\mathbf{L}(G)$ and $\mathbf{L}(H)$ on the same set of vertices. We have:

$$(1 - \epsilon)\mathbf{L}_{schur}(G) \preceq \mathbf{L}_{schur}(H) \preceq (1 + \epsilon)\mathbf{L}_{schur}(G)$$

The following fact is crucial in relating energy of the Schur complement to that of the original graph. It can be viewed as an instance of the Dirichlet Principle (see [DS84], page 64), which states that enforcing intermediate voltages other than the naturally occurring ones only increases total power. For completeness proof of it is included in Appendix C

Lemma B.2

$$\mathbf{x}^T \mathbf{L}_{schur}(G) \mathbf{x} = \min_{\mathbf{y}} \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix}^T \mathbf{L}(G) \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix}$$

Proof of Lemma B.1: We first show lower bound on $\mathbf{L}_{schur}(H)$. For a fixed vector \mathbf{x} , let \mathbf{y} the vector given by Lemma B.2 for $\mathbf{L}(H)$. Then we get:

$$\begin{aligned} (1 - \epsilon)\mathbf{x}^T \mathbf{L}_{schur}(G) \mathbf{x} &\leq (1 - \epsilon) \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix}^T \mathbf{L}(G) \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix} \\ &\leq \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix}^T \mathbf{L}(H) \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix} \\ &= \mathbf{x}^T \mathbf{L}_{schur}(H) \mathbf{x} \end{aligned}$$

For the upper bound on $\mathbf{L}_{schur}(H)$, once again let \mathbf{y} be given by Lemma B.2 for $\mathbf{L}(G)$ and \mathbf{x} . This gives:

$$\begin{aligned} (1 + \epsilon)\mathbf{x}^T \mathbf{L}_{schur}(G) \mathbf{x} &= (1 + \epsilon) \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix}^T \mathbf{L}(G) \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix} \\ &\geq \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix}^T \mathbf{L}(H) \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix} \\ &\geq \mathbf{x}^T \mathbf{L}_{schur}(H) \mathbf{x} \end{aligned}$$

■

We can also establish bounds relating the spectrum of \mathbf{L}_{schur} to the spectrum of \mathbf{L} .

Lemma B.3 Let $\mathbf{L}_{schur} = \text{SCHURCOMPLEMENT}(\mathbf{L}, V_{intr})$, then:

$$\begin{aligned} \lambda_2(\mathbf{L}_{schur}) &\geq \lambda_2(\mathbf{L}) \\ \lambda_n(\mathbf{L}_{schur}) &\leq n\lambda_n(\mathbf{L}) \end{aligned}$$

Proof of it is given in Appendix C.

Recall from Definition 2.6 that the Schur complement can be obtained by partitioning \mathbf{L} into blocks on the boundary and inner vertices:

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_{intr} & \mathbf{L}_{mid} \\ \mathbf{L}_{mid}^T & \mathbf{L}_{bdry} \end{bmatrix}$$

And evaluating $\mathbf{L}_{schur} = \mathbf{L}_{bdry} - \mathbf{L}_{mid}^T \mathbf{L}_{intr}^{-1} \mathbf{L}_{mid}$. Note that \mathbf{L}_{intr} is a graph Laplacian on the inner vertices with additional diagonal entries. This places it within the class of symmetric diagonally dominant (SDD) matrices. Furthermore, due to the graph being non-connected it is strictly SDD. We first show that the solves involving \mathbf{L}_{intr} can be done using faster SDD linear system solvers. We use the following result by Spielman and Teng on nearly-linear time solvers for such systems:

Lemma B.4 [*ST04, ST06, KMP10b, KMP11a*]. *Given a strictly SDD matrix of the form \mathbf{A} , and an error parameter δ , there is a symmetric linear operator \mathbf{B} such that*

$$(1 - \delta)\mathbf{A}^{-1} \preceq \mathbf{B} \preceq (1 + \delta)\mathbf{A}^{-1}$$

Furthermore, for any vector \mathbf{x} , $\mathbf{B}\mathbf{x}$ can be evaluated in time $\tilde{O}(m \log(1/\delta))$ where m is the number of non-zero entries in \mathbf{A} .

A direct approach is to apply the linear operator corresponding to \mathbf{L}_{bdry}^{-1} to \mathbf{L}_{mid} . As this is done one column at a time, we use $\mathbf{L}_{mid}[:, i]$ to denote the i -th column of \mathbf{L}_{mid} . Pseudocode for performing approximate Schur complements is given in Algorithm 4

Algorithm 4 Approximate Schur Complement Using SDD Linear System Solvers

APPROXSCHUR

Input: Graph $G = (V, E, \mathbf{w})$ with corresponding Laplacian \mathbf{L} , boundary vertices V_{bdry} . Spectrum bound κ such that $\lambda_n(\mathbf{L}) \leq \kappa \lambda_2(\mathbf{L})$, error parameter ϵ .

Output: Approximate Schur complement $\tilde{\mathbf{L}}_{schur}$.

- 1: Let V_{intr} denote the interior vertices, $V \setminus V_{bdry}$
 - 2: $\delta \leftarrow 2\epsilon n^{-1} \kappa^{-1}$
 - 3: **for** $i = 1 \dots |V_{bdry}|$ **do**
 - 4: $\mathbf{Y}[:, i] \leftarrow \text{SOLVE}(\mathbf{L}_{intr}, \mathbf{L}_{mid}[:, i], \delta)$
 - 5: **end for**
 - 6: $\tilde{\mathbf{L}}'_{schur} \leftarrow \mathbf{L}_{bdry} - \mathbf{L}_{mid}^T \mathbf{Y}$
 - 7: Let $\tilde{\mathbf{L}}_{schur}$ be $\tilde{\mathbf{L}}'_{schur}$ with positive off-diagonal entries replaced by 0s and diagonal entries readjusted to weighted degrees
 - 8: **return** $\tilde{\mathbf{L}}_{schur}$
-

Note that since the linear operator corresponding to SOLVE is symmetric, $\tilde{\mathbf{L}}'_{schur}$ is symmetric. However, SOLVE only produces an approximation to the inverse. We next show that setting δ to $\mathbf{poly}(n^{-1}, U(\mathbf{w})^{-1})$ suffices for approximating the Schur complement. We start with rough bounds on the maximum weight based on the eigenvalues.

Lemma B.5 *Let G be a graph with corresponding graph Laplacian \mathbf{L} . The maximum weight of an edge in L is at most $2\lambda_n$, and for any choice of boundary vertices, $\mathbf{L}_{bdry} \preceq 2n\lambda_n \mathbf{I}$.*

Proof By the Courant-Fisher theorem applied to the vector with 1 on one end and -1 on the other, we get that its weight is at most $2\lambda_n$.

Since \mathbf{L}_{bdry} is formed by taking a subset of the edges and adding weighted degrees to the diagonal, we have:

$$\mathbf{L}_{bdry} \preceq \mathbf{L} + n\lambda_n \mathbf{I} \preceq 2n\lambda_n \mathbf{I}$$

■

This allows us to prove the multiplicative error guarantee.

Lemma B.6 *Given a graph Laplacian \mathbf{L} for a graph G with n vertices and m edges, along with spectral bound κ such that $\lambda_n(\mathbf{L}) \leq \kappa\lambda_2(\mathbf{L})$, and error bound $0 < \epsilon < 1/2$. Let \mathbf{L}_{schur} be the exact Schur complement of (G, V_{intr}) . Then $\text{APPROXSCHUR}(G, V_{bdry}, \kappa, \epsilon)$ in $\tilde{O}(|V_{bdry}|m \log(\kappa\epsilon^{-1}))$ time outputs a matrix $\tilde{\mathbf{L}}_{schur}$ such that:*

$$(1 - \epsilon)\mathbf{L}_{schur} \preceq \tilde{\mathbf{L}}_{schur} \preceq (1 + \epsilon)\mathbf{L}_{schur}$$

Proof

We only show the LHS inequality as the RHS side follows similarly. Let \mathbf{B} be the linear operator generated by SOLVE. Then by the guarantees given in Lemma B.4 we have $\mathbf{B} \preceq (1 + \delta)\mathbf{L}_{intr}^{-1}$. This in turn gives $\mathbf{L}_{mid}^T \mathbf{B} \mathbf{L}_{mid} \preceq (1 + \delta)\mathbf{L}_{mid}^T \mathbf{L}_{intr}^{-1} \mathbf{L}_{mid}$ and:

$$\begin{aligned} \tilde{\mathbf{L}}_{schur} &= \mathbf{L}_{bdry} - \mathbf{L}_{mid}^T \mathbf{B} \mathbf{L}_{mid} \\ &\succeq \mathbf{L}_{bdry} - (1 + \delta)\mathbf{L}_{mid}^T \mathbf{L}_{intr}^{-1} \mathbf{L}_{mid} \\ &= \mathbf{L}_{schur} - \delta\mathbf{L}_{mid}^T \mathbf{L}_{intr}^{-1} \mathbf{L}_{mid} \end{aligned}$$

Since $\mathbf{L}_{bdry} - \mathbf{L}_{mid}^T \mathbf{L}_{intr}^{-1} \mathbf{L}_{mid}$ is the Schur complement and is positive semi-definite, we may use $\delta\mathbf{L}_{bdry}$ as an upper bound for the difference. Specifically we can use the bound on \mathbf{L}_{bdry} given above. Combining it with $\kappa_2(\mathbf{L}_{schur}) \leq \lambda_2(\mathbf{L})$ from Lemma B.3 finishes the bound:

$$\begin{aligned} \tilde{\mathbf{L}}_{schur} &\succeq \mathbf{L}_{schur} - \delta 2n\lambda_n(\mathbf{L})\mathbf{I} \\ &\succeq \mathbf{L}_{schur} - \epsilon\lambda_2(\mathbf{L})\mathbf{I} \\ &\succeq \mathbf{L}_{schur} - \epsilon\lambda_2(\mathbf{L}_{schur})\mathbf{I} \\ &\succeq (1 - \epsilon)\mathbf{L}_{schur} \end{aligned}$$

We now bound the running time. The inner loop consists of $|V_{bdry}|$ calls to SOLVE, each taking at most $\tilde{O}(m \log \delta)$ to evaluate. These steps combine for a total of $\tilde{O}(|V_{bdry}|m \log(\kappa\epsilon^{-1}))$. Also, \mathbf{L}_{mid}^T is a sparse matrix with $O(m)$ non-zero entries corresponding to the edges. So multiplying it against $\mathbf{B}\mathbf{L}_{mid}$, which has $|V_{bdry}|$ columns can be done in $O(|V_{bdry}|m)$ time. Combining these two steps gives the overall bound. ■

The requirements for APPROXSCHUR are stated in terms of spectrum to ease the use of it within recursive routines later in this section. We make use of the following lemmas to convert between rough bounds on edge weights and spectrum.

Lemma B.7 *Let $G = (V, E, \mathbf{w})$ be a connected graph with corresponding graph Laplacian \mathbf{L} . We can find in $O(m)$ time eigenvalue bounds λ_{\min} and λ_{\max} such that $\lambda_{\min} \leq \lambda_2(\mathbf{L}) \leq \lambda_n(\mathbf{L})\lambda_{\max} \leq n^3 U(\mathbf{w})\lambda_{\min}$.*

Proof Let \mathbf{w}_{\min} and \mathbf{w}_{\max} denote the minimum and maximum weight in G . Since G is connected, it has a spanning tree consisting of edges of weight \mathbf{w}_{\min} , giving $\lambda_{\min} \geq \frac{1}{n^2}\mathbf{w}_{\min}$.

Also, since all edge weights of G are at most \mathbf{w}_{\max} , we have $\mathbf{L} \preceq \mathbf{w}_{\max} \mathbf{L}(K_n)$ where K_n is the complete graph on n vertices. Therefore $\lambda_{\max} = n\mathbf{w}_{\max}$ suffices and its relation with λ_{\min} follows from $\mathbf{w}_{\max} \leq U\mathbf{w}_{\min}$. \blacksquare

Lemma B.8 *Given a graph $G = (V, E, \mathbf{w})$ and bound $\lambda_{\min} \leq \lambda_2(\mathbf{L})$. We can compute in $O(m)$ time another graph $G' = (V, E, \mathbf{w}')$ on the same edge set such that the minimum edge weight in G' is $\frac{1}{n^2}\lambda_{\min}$ and:*

$$\mathbf{L} \preceq \mathbf{L}' \preceq (1 + \frac{1}{n})\mathbf{L}$$

Proof We have $\frac{\lambda_{\min}}{n^2}\mathbf{L}(K_n) \preceq \frac{1}{n}\mathbf{L}$ where K_n is the complete graph on n vertices. Therefore adding $\frac{\lambda_{\min}}{n^2}$ to each edge of G gives G' . \blacksquare

Once we obtain the approximate inverse, its edge count can be reduced using spectral sparsifiers. The guarantees of the sparsification algorithm, first introduced by Spielman and Srivastava [SS08] is:

Lemma B.9 [SS08, KL11] *There is a routine SPARSIFY such that given a graph Laplacian \mathbf{L}_G on n vertices with m edges and a parameter ϵ . SPARSIFY(G, ϵ) outputs in $\tilde{O}(m \log(\epsilon^{-1}))$ time, a graph Laplacian \mathbf{L}_H with $\tilde{O}(n\epsilon^{-2})$ edges such that $(1 - \epsilon)\mathbf{L}_G \preceq \mathbf{L}_H \preceq (1 + \epsilon)\mathbf{L}_G$.*

Combining spectral sparsification with APPROXSCHUR completes the proof of lemma 3.6. The pseudocode of ONESTEPVERTEXSPARSIFY is given in Algorithm 5

Algorithm 5 One Step Spectral Vertex Sparsification

ONESTEPVERTEXSPARSIFY

Input: Graph Laplacian \mathbf{L} corresponding to $G = (V, E, \mathbf{w})$ with vertices V and boundary vertices V_{bdry} . allowable error per step ϵ .

Output: Approximate Schur complement $\tilde{\mathbf{L}}_{schur}$.

- 1: Let $\lambda_{\min} = \frac{1}{n^2} \min_e \mathbf{w}(e)$
 - 2: Let $\lambda_{\max} = n \max_e \mathbf{w}(e)$
 - 3: Let $\kappa = \lambda_{\max}/\lambda_{\min}$
 - 4: $\tilde{\mathbf{L}}_{schur} \leftarrow \text{APPROXSCHUR}(G, V_{bdry}, \kappa, \epsilon/3)$
 - 5: $\tilde{\mathbf{L}}'_{schur} \leftarrow \text{SPARSIFY}(\tilde{\mathbf{L}}_{schur}, \epsilon/3)$
 - 6: Add $\frac{1}{n^2}\lambda_{\min}$ to the weight of all edges present in $\tilde{\mathbf{L}}'_{schur}$ to obtain $\tilde{\mathbf{L}}''_{schur}$
 - 7: **return** $\tilde{\mathbf{L}}''_{schur}$
-

Proof of Lemma 3.6: By Lemma B.7 we have the that λ_{\min} , λ_{\max} and κ are bounds for eigenvalues. Lemma B.9 gives that $\tilde{\mathbf{L}}''_{schur}$ has $\tilde{O}(n\epsilon^{-2})$ edges and:

$$(1 - \epsilon/3)\tilde{\mathbf{L}}_{schur} \preceq \tilde{\mathbf{L}}'_{schur} \preceq (1 + \epsilon/3)\mathbf{L}_{schur}$$

Combining with Lemma B.6 gives that $\tilde{\mathbf{L}}'_{schur}$ meets the requirements of an ϵ -spectral vertex sparsifier. By Lemmas B.1 and B.2, 2κ is sufficient as spectrum bound for $\tilde{\mathbf{L}}''_{schur}$. Therefore using Lemma B.8 we have $\tilde{\mathbf{L}}'' \preceq (1 + \frac{1}{n})\tilde{\mathbf{L}}'_{schur} \preceq (1 + \epsilon)\mathbf{L}_{schur}$. Furthermore, the edge weights in \mathbf{L}_{schur} are at least $\frac{1}{n^2}\lambda_{\min}$ and at most $O(n\lambda_{\max})$, giving a bound of $O(n^5 U(\mathbf{w}))$. As SPARSIFY runs in nearly-linear time and $\tilde{\mathbf{L}}_{schur}$ has at most $|V_{bdry}|^2 \leq |V_{bdry}|m$ edges, APPROXSCHUR is the bottleneck in the total run time. \blacksquare

We now move on to our multilevel recursive sparsification algorithm for graphs where we have access to a full separator tree. Pseudocode of our algorithm is shown in Algorithm 6

Algorithm 6 Recursive Spectral Vertex Sparsification for Well-Separated Graphs

VERTEXSPARSIFY

Input: Node in separator tree \mathcal{S} , graph G , with separator S and children $G^{(1)} = (V^{(1)}, E^{(1)})$ and $G^{(2)} = (V^{(2)}, E^{(2)})$. Graph Laplacian \mathbf{L} corresponding to G with vertices V and boundary vertices V_{bdry} . Spectrum bound κ such that $\lambda_n(\mathbf{L}) \leq \kappa \lambda_2(\mathbf{L})$, allowable error per step ϵ .

Output: Approximate Schur complement $\tilde{\mathbf{L}}_{schur}$.

```

1: for  $i = \{1, 2\}$  do
2:    $V_{bdry}(G^{(i)}) \leftarrow (V^{(i)} \cap V_{bdry}) \cup S$ 
3:    $\tilde{\mathbf{L}}_{schur}^{(i)} \leftarrow \text{VERTEXSPARSIFY}(\mathbf{L}^{(i)}, V_{bdry}(G^{(i)}), \kappa, \epsilon)$ 
4: end for
5:  $\tilde{\mathbf{L}}'_{schur} \leftarrow \tilde{\mathbf{L}}_{schur}^{(1)} + \tilde{\mathbf{L}}_{schur}^{(2)}$ 
6:  $\tilde{\mathbf{L}}''_{schur} \leftarrow \text{APPROXSCHUR}(\tilde{\mathbf{L}}'_{schur}, V_{bdry}, 2^{\log_{20/19} n} \kappa, \epsilon/3)$ 
7:  $\tilde{\mathbf{L}}_{schur} \leftarrow \text{SPARSIFY}(\tilde{\mathbf{L}}''_{schur}, \epsilon/3)$ 
8: return  $\tilde{\mathbf{L}}_{schur}$ 

```

In order to provide error guarantees for VERTEXSPARSIFY, we relate rough spectral bounds between. This in turn allows applications of Lemma B.6 in an inductive fashion. These guarantees can be summarized as follows:

Lemma B.10 *Let G be a graph with n vertices and m edges such that $\lambda_n(G) \leq \kappa \lambda_2(G)$, Given a partition of the vertices V into boundary and interior vertices such that $|V_{bdry}| \leq \sqrt{n}$, a $9/10$ separator tree \mathcal{S} , and an error parameter ϵ .*

Then $\tilde{\mathbf{L}}'_{schur}$ as defined on Line 5 of VERTEXSPARSIFY satisfies:

$$\lambda_n(\tilde{\mathbf{L}}'_{schur}) \leq 2^{\log_{20/19} n} \kappa \lambda_2(\tilde{\mathbf{L}}'_{schur})$$

And $\text{VERTEXSPARSIFY}(\mathbf{L}, V_{bdry}, \kappa, \epsilon)$ returns $\tilde{\mathbf{L}}_{schur}$ with $\tilde{O}(n\epsilon^{-2})$ edges such that:

$$(1 - \epsilon)^{\log_{20/19} n} \mathbf{L}_{schur} \preceq \tilde{\mathbf{L}}_{schur} \preceq (1 + \epsilon)^{\log_{20/19} n} \mathbf{L}_{schur}$$

Proof The proof is by induction on n . Since $|S|$ has size at most $O(\sqrt{n})$ we have that $|V_{bdry}(G^{(1)})|, |V_{bdry}(G^{(2)})| \leq O(\sqrt{n})$. Then since the two graphs that we recurse on have size at most $9/10n + O(\sqrt{n}) \leq 19/20n$, by the induction hypothesis we have:

$$(1 - \epsilon)^{\log_{20/19} n-1} \mathbf{L}_{schur}^{(i)} \preceq \tilde{\mathbf{L}}_{schur}^{(i)} \preceq (1 + \epsilon)^{\log_{20/19} n-1} \mathbf{L}_{schur}^{(i)}$$

Furthermore, we can compute an approximate Schur complement of this sum, only introducing an extra error of $(i-1)/2 \log n \epsilon$. Let \mathbf{L}'_{schur} be the (exact) Schur complement of $(G, V_{intr}) \setminus S$. Then invoking the induction hypothesis and summing over the two pieces gives:

$$(1 - \epsilon)^{\log_{20/19} n-1} \mathbf{L}'_{schur} \preceq \tilde{\mathbf{L}}'_{schur} \preceq (1 + \epsilon)^{\log_{20/19} n-1} \mathbf{L}'_{schur}$$

Combining this with the relation of the spectrum of \mathbf{L}_{schur} and \mathbf{L} gives:

$$\begin{aligned}
& \lambda_n(\tilde{\mathbf{L}}'_{schur}) \\
& \leq (1 + \epsilon)^{\log_{20/19} n} \lambda_n(\mathbf{L}_{schur}) \\
& \leq (1 + \epsilon)^{\log_{20/19} n} n \lambda_n(\mathbf{L}) \\
& \leq (1 + \epsilon)^{\log_{20/19} n} n \kappa \lambda_2(\mathbf{L}) \\
& \leq (1 + \epsilon)^{\log_{20/19} n} (1 - \epsilon)^{-\log_{20/19} n} n \kappa \lambda_2(\tilde{\mathbf{L}}'_{schur}) \\
& \leq 2^{\log_{20/19} n} n \kappa \lambda_2(\tilde{\mathbf{L}}'_{schur}) \\
& \quad \text{By assumption that } \epsilon \leq 1/10,
\end{aligned}$$

Therefore running APPROXSCHUR with spectrum bound of $2^{\log_{20/19} n} n \kappa$ satisfies the input requirements for in Lemma B.1. Let \mathbf{L}''_{schur} in turn be the exact Schur complement of removing S from $\tilde{\mathbf{L}}'_{schur}$, Lemma B.1 then gives that the approximation guarantee still holds.

$$(1 - \epsilon)^{(\log_{20/19} n) - 1} \mathbf{L}_{schur} \preceq \mathbf{L}''_{schur} \preceq (1 + \epsilon)^{(\log_{20/19} n) - 1} \mathbf{L}_{schur}$$

By Lemma B.6, $\tilde{\mathbf{L}}''_{schur}$ satisfies:

$$(1 - \epsilon/3) \mathbf{L}''_{schur} \preceq \tilde{\mathbf{L}}''_{schur} \preceq (1 + \epsilon/3) \mathbf{L}''_{schur}$$

Lemma B.9 gives an additional error of $1 \pm \epsilon/3$ from the sparsification, as well as the bound on edge count. Including this error gives:

$$(1 - \epsilon) \mathbf{L}''_{schur} \preceq \tilde{\mathbf{L}}_{schur} \preceq (1 + \epsilon) \mathbf{L}''_{schur}$$

Combining this with the bound between \mathbf{L}_{schur} and \mathbf{L}''_{schur} shows the inductive hypothesis for n as well. ■

We now turn our attention to the running time, which is dominated by the calls to APPROXSCHUR.

Lemma B.11 *Let G be a graph with n vertices and m edges such that $\lambda_n(G) \leq \kappa \lambda_2(G)$, Given a partition of the vertices V into boundary and interior vertices such that $|V_{bdry}| \leq \sqrt{n}$, a 9/10 separator tree \mathcal{S} , and an error parameter ϵ , VERTEXSPARSIFY($\mathbf{L}, V_{bdry}, \kappa, \epsilon$) terminates in $\tilde{O}((m + |V_{bdry}|^2) \epsilon^{-2} \log(\kappa))$ time*

Proof

Since $\tilde{\mathbf{L}}_{schur}^{(1)}$ and $\tilde{\mathbf{L}}_{schur}^{(1)}$ combine for $|V_{bdry}(G^{(1)})| + |V_{bdry}(G^{(2)})| \leq 2(|V_{bdry}| + |S|)$ vertices, $\tilde{\mathbf{L}}'_{schur}$ has at most $\tilde{O}((|V_{bdry}| + |S|) \epsilon^{-2})$ edges by the induction hypothesis. So by Lemma B.6, the running time of this call is bounded by $\tilde{O}((|V_{bdry}| + |S|) |V_{bdry}| \epsilon^{-2} \log(\kappa))$.

Therefore the total running time can be bounded by $\tilde{O}(\epsilon^{-2} \log(\kappa))$ times the total value of $(|V_{bdry}| + |S|) |V_{bdry}|$ summed over all the recursive calls. We follow the proof of Theorem 2 of [LRT79], which bounds the fill of generalized nested dissection. We also use $f(l, n)$ to denote the maximum sum of $(|V_{bdry}| + |S|) |V_{bdry}|$ over the recursive calls starting from a graph with n vertices, l of which are on the boundary.

Let c_0 be the constant in the 9/10 separator tree where the separator size is at most $c_0 \sqrt{n}$. Then the recursive calls guarantee that:

- If $n \leq n_0$ for some absolute constants C and n_0 , then $f(l, n) \leq C$.

- Otherwise, $f(l, n) \leq l(l + c_0 n) + f(l_1, n_1) + f(l_2, n_2)$ for l_1, l_2, n_1, n_2 that satisfy:

$$\begin{aligned} l_1 + l_2 &\leq l + 2c_0\sqrt{n} \\ n_1 + n_2 &\leq n + c_0\sqrt{n} \\ n_1, n_2 &\leq 9/10n + c_0\sqrt{n} \end{aligned}$$

Then we show by induction on n that there exists an absolute constants c_1 and c_2 such that:

$$f(l, n) \leq c_1 l^2 \log n + c_2 n \log^2 n$$

The base case of $n \leq n_0$ follows from C being a constant. For the inductive case the RHS is bounded by:

$$\begin{aligned} &c_1 l_1^2 \log n_1 + c_2 n_1 \log^2 n_1 \\ &\quad + c_1 l_2^2 \log n_2 + c_2 n_2 \log^2 n_2 + l(l + c_0\sqrt{n}) \\ &\leq c_1 \log(\max(n_1, n_2))(l_1^2 + l_2^2) \\ &\quad + c_2 \log(\max(n_1, n_2))(n_1 + n_2) + l(l + c_0\sqrt{n}) \\ &\leq c_1 (\log(n) - \log(10/9))(l_1 + l_2)^2 \\ &\quad + c_2 (\log(n) - \log(10/9))^2 (n_1 + n_2) + l(l + c_0\sqrt{n}) \\ &\leq c_1 (\log(n) - \log(10/9))(l + 2c_0\sqrt{n})^2 \\ &\quad + c_2 (\log(n) - \log(10/9))^2 (n + c_0\sqrt{n}) + l(l + c_0\sqrt{n}) \end{aligned}$$

By the arithmetic-geometric mean inequality we have $l \cdot c_0\sqrt{n} \leq (l^2 + c_0^2 n)/2$. Making this substitution gives:

$$\begin{aligned} &\leq c_1 (\log(n) - \log(10/9))(l^2 + 8c_0^2 n) \\ &\quad + c_2 (\log(n) - \log(10/9))^2 (n + c_0\sqrt{n}) + 2l^2 + c_0^2 n \\ &\leq (c_1 \log(n) - c_1 \log(10/9) + 2)l^2 \\ &\quad + (c_2 (\log(n) - \log(10/9))^2 + 8c_1 \log(n)c_0^2 + c_0^2) n \\ &\quad + c_2 \log^2(n) c_0\sqrt{n} \end{aligned}$$

By an appropriate choice of n_0 we can ensure that $\log^2(n)\sqrt{n} \leq n$, giving a simpler upper bound of:

$$\begin{aligned} &\leq (c_1 \log(n) - c_1 \log(10/9) + 2)l^2 \\ &\quad + (c_2 (\log(n) - \log(10/9))^2 + 8c_1 \log(n)c_0^2 + 2c_0^2) n \end{aligned}$$

By choosing $c_1 = 2/\log(10/9)$, we get the coefficient on l^2 to be at most $c_1 \log n$. Then since $c_2 \log(n)^2 - c_2 (\log(n) - \log(10/9))^2 = 2c_2 \log(10/9) - \log(10/9)^2$, and c_1, c_0 are fixed constants, a sufficiently large constant for c_2 ensures the coefficient in front of n is also bounded by $c_2 \log n$. Combining with the extra term of $\tilde{O}(\epsilon^{-2} \log(\kappa))$ gives the first term in the running time bound. An additional term of m is needed to account for the cost of the initial sparsification. ■

Proof of Lemma 3.7: The conversion between guarantees on edge weight ratio $U(\mathbf{w})$ and spectrum can be handled in the same way as in Lemma 3.6. Combining Lemmas B.10 and B.11, and adjusting

parameters by replacing ϵ with $\epsilon/2 \log_{20/19} n$ gives the required bounds. ■

C Proofs of Properties of Schur Complement

Proof of Lemma 5.1:

Let the vectors \mathbf{x} be the solution to the solution to $\mathbf{L}\mathbf{x} = \mathbf{d}$. Partitioning \mathbf{x} into \mathbf{x}_{bdry} and \mathbf{x}_{intr} gives:

$$\begin{bmatrix} \mathbf{L}_{intr} & \mathbf{L}_{mid} \\ \mathbf{L}_{mid}^T & \mathbf{L}_{bdry} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{intr} \\ \mathbf{x}_{bdry} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{d}_{bdry} \end{bmatrix}$$

The equality given by the first set of blocks can be rewritten as:

$$\begin{aligned} \mathbf{L}_{intr}\mathbf{x}_{intr} + \mathbf{L}_{mid}\mathbf{x}_{bdry} &= \mathbf{0} \\ \mathbf{x}_{intr} &= -\mathbf{L}_{intr}^{-1}\mathbf{L}_{mid}\mathbf{x}_{bdry} \end{aligned}$$

Substituting this into the second set of blocks gives:

$$\begin{aligned} -\mathbf{L}_{mid}^T\mathbf{L}_{intr}^{-1}\mathbf{L}_{mid}\mathbf{x}_{bdry} + \mathbf{L}_{bdry}\mathbf{x}_{bdry} &= \mathbf{d}_{bdry} \\ (\mathbf{L}_{bdry} - \mathbf{L}_{mid}^T\mathbf{L}_{intr}^{-1}\mathbf{L}_{mid})\mathbf{x}_{bdry} &= \mathbf{d}_{bdry} \\ \mathbf{x}_{bdry} &= \mathbf{L}_{schur}\mathbf{d}_{bdry} \end{aligned}$$

Which if we substitute into $\mathbf{d}^T\mathbf{L}^+\mathbf{d}$ and combining with the fact that \mathbf{d} is only non-zero on the boundary vertices gives:

$$\mathbf{d}^T\mathbf{L}^+\mathbf{d} = \mathbf{d}_{bdry}^T\mathbf{x}_{bdry} = \mathbf{d}_{bdry}^T\mathbf{L}_{schur}^+\mathbf{d}_{bdry}(i)$$

■ Lemma 5.1

Proof of Lemma B.2:

If we treat \mathbf{x} as a constant, then the expression to be minimized can be written as:

$$\begin{aligned} &\begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix}^T \mathbf{L}(G) \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix}^T \begin{bmatrix} \mathbf{L}_{intr} & \mathbf{L}_{mid} \\ \mathbf{L}_{mid}^T & \mathbf{L}_{bdry} \end{bmatrix} \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix} \\ &= \mathbf{y}^T\mathbf{L}_{intr}\mathbf{y} + 2\mathbf{y}^T\mathbf{L}_{mid}\mathbf{x} + \mathbf{x}^T\mathbf{L}_{bdry}\mathbf{x} \end{aligned}$$

The first two terms can be viewed as a quadratic in \mathbf{y} , and is therefore minimized at $\bar{\mathbf{y}} = -\mathbf{L}_{intr}^{-1}\mathbf{L}_{mid}\mathbf{x}$. Substituting this into the expression above gives:

$$\begin{aligned} &\min_{\mathbf{y}} \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix}^T \mathbf{L}(G) \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix} \\ &= \bar{\mathbf{y}}^T\mathbf{L}_{intr}\bar{\mathbf{y}} + 2\bar{\mathbf{y}}^T\mathbf{L}_{mid}\mathbf{x} + \mathbf{x}^T\mathbf{L}_{bdry}\mathbf{x} \\ &= \mathbf{x}^T\mathbf{L}_{bdry}\mathbf{x} - \mathbf{x}^T\mathbf{L}_{mid}^T\mathbf{L}_{intr}^{-1}\mathbf{L}_{mid}\mathbf{x} \\ &= \mathbf{x}^T(\mathbf{L}_{bdry} - \mathbf{L}_{mid}^T\mathbf{L}_{intr}^{-1}\mathbf{L}_{mid})\mathbf{x} \\ &= \mathbf{x}^T\mathbf{L}_{schur}\mathbf{x} \end{aligned}$$

■ Lemma B.2

Proof of Lemma B.3:

Let \mathbf{x} be any vector on the support of \mathbf{L}_{schur} . Then by Lemma B.2 there exist a vector \mathbf{y} such that:

$$\mathbf{x}^T \mathbf{L}_{schur}(G) \mathbf{x} = \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix}^T \mathbf{L}(G) \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix} \quad (3.1)$$

Furthermore, all entries of \mathbf{y} are between the minimum and maximum entries of \mathbf{x} , giving:

$$\mathbf{x}^T \mathbf{x} \leq \mathbf{x}^T \mathbf{x} + \mathbf{y}^T \mathbf{y} \leq n \mathbf{x}^T \mathbf{x} \quad (3.2)$$

The bounds then follow by setting \mathbf{x} to eigenvectors of \mathbf{L}_{schur} and applying the Courant-Fisher theorem on \mathbf{L} . \blacksquare

D Returning a Small Cut

We now show that if GROUPEDFLOW returns “fail”, then a small cut can also be found. We begin by stating one omitted aspect of the electrical flow algorithm as given in Theorem 2.3 of [CKM⁺11].

Lemma D.1 ELECTRICALFLOW can be modified to return a set of vertex potentials $\tilde{\phi}$ such that the energy given by the potentials $\mathcal{E}_{\tilde{\phi}}(\mathbf{r})$ is at most 1, and $\mathbf{d}^T \tilde{\phi}$ is at least $(1 - \delta)\mathcal{E}(\tilde{\mathbf{f}})$.

These vertex potentials are approximate versions of the voltages corresponding to the electrical flow. Since the vertex sparsified graph is approximately the same as the graph given as input to GROUPEDFLOW, we show that this can be extended to a full set of potentials on vertices of the original graph, which in turn leads to a small cut.

Theorem D.2 If GROUPEDFLOW returns “fail”, we can find in $\tilde{O}(m)$ time a set of vertex potentials ϕ such that:

1. $\sum_{e=uv \in E} \mathbf{u}(e) |\phi_u - \phi_v| \leq 1$
2. $\mathbf{d}^T \phi \geq 1 - 10\epsilon$

Proof

By Lemma D.1, in this situation a set of vertex potentials $\tilde{\phi}_{schur}$ on \tilde{G}_{schur} such that: $\tilde{\phi}_{schur}^T \tilde{\mathbf{L}}_{schur} \tilde{\phi}_{schur} \leq 1$ and $\mathbf{d}_{bdry}^T \tilde{\phi}_{schur} \geq \mu$ where $\mu = \sum_{1 \leq i \leq k} \mathbf{w}_{grp}(i)$. Let the weights among edges given ORACLE be \mathbf{w} and the weights set in GROUPEDFLOW be \mathbf{w}_{grp} . Furthermore, for $e \in E(G^{(i)})$ we use $\mathbf{w}_{combined}(e)$ to denote combined weight on e that does not account for its capacity:

$$\begin{aligned} \mathbf{w}_{combined}(e) &= \mathbf{u}(e)^2 \mathbf{r}(e) \\ &= (\mathbf{w}_{grp}(i) + \frac{\epsilon}{k} \mu) \cdot \left(\frac{\mathbf{w}(e)}{\mathbf{w}(G^{(i)})} + \frac{\epsilon}{|E(G^{(i)})|} \right) \end{aligned}$$

Consider each component i , let $\tilde{\phi}_{schur}(i)$ denote the restriction of $\tilde{\phi}_{schur}$ to the vertices on $V_{bdry}(G^{(i)})$. Since $\tilde{\mathbf{L}}_{schur}(G^{(i)}) \preceq (1 + \epsilon) \mathbf{L}_{schur}(G^{(i)})$, we have:

$$\tilde{\phi}_{schur}(i)^T \mathbf{L}_{schur}(G^{(i)}) \tilde{\phi}_{schur}(i) \leq (1 + \epsilon) \tilde{\phi}_{schur}(i)^T \tilde{\mathbf{L}}_{schur}(G^{(i)}) \tilde{\phi}_{schur}(i)$$

Therefore by Lemma B.2, it's possible to extend $\tilde{\phi}_{schur}(i)$ onto the interior vertices of $G^{(i)}$ to obtain $\tilde{\phi}(i)$ such that:

$$\tilde{\phi}(i)^T \mathbf{L}(G^{(i)}) \tilde{\phi}(i) \leq (1 + \epsilon) \tilde{\phi}_{schur}(i)^T \tilde{\mathbf{L}}_{schur}(G^{(i)}) \tilde{\phi}_{schur}(i)$$

Summing over all clusters gives a vector $\tilde{\phi}$ such that:

$$\begin{aligned} \tilde{\phi}^T \mathbf{L} \tilde{\phi} &= \sum_{1 \leq i \leq k} \tilde{\phi}(i)^T \mathbf{L}(G^{(i)}) \tilde{\phi}(i) \\ &\leq \sum_{1 \leq i \leq k} \tilde{\phi}_{schur}(i)^T \tilde{\mathbf{L}}_{schur}(G^{(i)}) \tilde{\phi}_{schur}(i) \\ &= (1 + \epsilon) \tilde{\phi}_{schur}^T \tilde{\mathbf{L}}_{schur}(G^{(i)}) \tilde{\phi}_{schur} \\ &\leq 1 + \epsilon \end{aligned} \tag{4.3}$$

Furthermore since \mathbf{d} is 0 in all interior vertices, we have:

$$\mathbf{d}^T \tilde{\phi} = \mathbf{d}_{bdry}^T \tilde{\phi}_{schur} \geq \mu$$

Then consider $\phi = \frac{1}{(1+10\epsilon)\mu}$. We have $\mathbf{d}^T \phi \geq 1 - 10\epsilon$, and from Equation 4.3 we can obtain:

$$\begin{aligned} \phi^T \mathbf{L} \phi &\leq \frac{1 + \epsilon}{(1 + 10\epsilon)\mu} \\ (1 + 8\epsilon)\mu(\phi^T \mathbf{L} \phi) &\leq 1 \end{aligned} \tag{4.4}$$

Substituting in the fact that the resistive weight of edge $e \in E(G^{(i)})$ in \mathbf{L} is $\mathbf{r}(e) = \frac{\mathbf{w}_{combined}(e)}{\mathbf{u}(e)^2}$ gives:

$$\begin{aligned} \phi^T \mathbf{L} \phi &= \sum_{1 \leq i \leq k} \sum_{e=uv \in E(G^{(i)})} \frac{(\phi(u) - \phi(v))^2}{\mathbf{r}(e)} \\ &= \sum_{e=uv \in E(G^{(i)})} \frac{\mathbf{u}(e)^2 (\phi(u) - \phi(v))^2}{\mathbf{w}_{combined}(e)} \end{aligned}$$

Also, we have that the total sum of resistive values among all the clusters is:

$$\begin{aligned}
& \sum_e \mathbf{w}_{combined}(e) \\
&= \sum_{1 \leq i \leq k} \sum_{e \in E(G^{(i)})} (\mathbf{w}_{grp}(i) + \frac{\epsilon}{k} \mu) \cdot \left(\frac{\mathbf{w}(e)}{\mathbf{w}(G^{(i)})} + \frac{\epsilon}{|E(G^{(i)})|} \right) \\
&= \sum_{1 \leq i \leq k} (\mathbf{w}_{grp}(i) + \frac{\epsilon}{k} \mu) \left(\sum_{e \in E(G^{(i)})} \frac{\mathbf{w}(e)}{\mathbf{w}(G^{(i)})} + \frac{\epsilon}{|E(G^{(i)})|} \right) \\
&= (1 + \epsilon) \sum_{1 \leq i \leq k} (\mathbf{w}_{grp}(i) + \frac{\epsilon}{k} \mu) \\
&\leq (1 + 3\epsilon) \mu
\end{aligned}$$

Substituting into Equation 4.4 gives:

$$\begin{aligned}
1 &\geq (1 + 8\epsilon) \mu (\phi^T \mathbf{L} \phi) \\
&\geq \left(\sum_e \mathbf{w}_{combined}(e) \right) \left(\sum_e \frac{\mathbf{u}(e)^2 (\phi(u) - \phi(v))^2}{\mathbf{w}_{combined}(e)} \right) \\
&\geq \left(\sum_e \mathbf{u}(e) |\phi(u) - \phi(v)| \right)^2
\end{aligned}$$

By the Cauchy-Schwarz inequality

Taking square root of both sides completes the proof. ■